Bound by

ABBEY BOOKBINDING & PRINTING
Unit 3 Gabalfa Workshops Excelsior Ind. Est. Cardiff CF14 3AY
Tel: (029) 2062 3290 Fax: (029) 2062 5420
Email: info@abbeybookbinding.co.uk
Web: www.abbeybookbinding.co.uk

MODELLING AND ANALYSING 3D BUILDING INTERIORS WITH THE DUAL HALF-EDGE DATA STRUCTURE

PAWEL BOGUSLAWSKI

A submission presented in partial fulfilment of the
requirements of the University of Glamorgan/Prifysgol Morgannwg
for the degree of Doctor of Philosophy

This research was supported by the
Ordnance Survey and
EPSRC funding of a New CASE award

March 2011

Abstract

While many systems and standards like CAD systems or CityGML permit the user to represent the geometry and the semantics of building interior models, their use for applications where spatial analysis and/or real-time modifications are required are limited since they lack the possibility to store topological relationships between the elements. In this thesis a new topological data structure, the dual half-edge (DHE) is presented. It permits the representation of the topology of building models with the interior included. It is based on the idea of simultaneously storing a graph in 3D space and its dual graph, and to link the two. Euler-type operators for incrementally constructing 3D models (for adding individual edges, faces and volumes to the model while updating the dual structure simultaneously), and navigation operators (for example to navigate from a given point to all the connected planes or polyhedra) are proposed. The DHE also permits the assigning of attributes to any element. This technique allows the handling of important query types and performs analysis based on the building structure, for example finding the nearest exterior exit to a given room, as in disaster management planning. As the structure is locally modifiable the model may be adapted whenever a particular pathway is no longer available. The proposed DHE structure adds significant analytic value to the increasingly popular CityGML model, and to the CAD field where the dual structure is of growing interest.

Acknowledgements

Most of all I would like to thank Chris Gold who was my supervisor. He triggered my love affair with real science which I want to continue. Chris was very patient answering my questions about our research (and many other things). When words could not express our minds we switched to a manual conversation – we waved our arms to explain the complexities of connections in our models in different dimensions (mainly 3D – to talk about 4D we need more arms to wave). I am grateful for Chris's support and encouragement. Working with Chris was a pleasure and a special privilege.

Many thanks to Hugo Ledoux who has helped me to improve the quality of my research. He gave me many ideas and asked questions which made me think. The result of our collaboration was a journal paper we wrote with Chris. I hope we will have an opportunity to work together in the future.

I want to thank Maciek Dakowicz and Rafal Goralski who persuaded Chris to take another student from Poland and who persuaded me to leave everything and go to Cardiff. Maciek also infected me with love of photography – he taught me a lot about taking pictures.

During the PhD project I went to several conferences. Organizing journeys, booking hotels and tickets, and all the paperwork was a piece of cake thanks to Pat Bayley from the Research

Office. She helped me every time I had a problem starting from my first visit when I barely spoke English. Thank you Pat!

The project would not have been possible without the financial support of the Ordnance Survey. They believed from the beginning that we would be successful. During our collaboration I had the pleasure to meet Dave Capstick and Mark Pendlington who were very encouraging and interested in the results of our research – we had several meetings and spent many hours talking about the project. I am very grateful for the time you dedicated to me.

My special thanks go to my wife Patrycja and my family for their encouragement and exemption from the housework while I wrote the thesis. Special thanks to Valerie and Chris Gold who were my family during the time I spent in Cardiff working on the PhD project – you will stay in my heart forever.

Table of Contents

1 Introduction

The management of spatial relations and spatial objects is important in a variety of disciplines. For example CAD systems have to be able to manipulate the boundaries of individual 'shells' in order to model the engineer's desired object. Modern medical tomography requires the reconstruction of irregular 'blob shaped' objects from sequences of closely spaced scans. Architecture requires the assembly of multiple parts to form a complete building. Geographic Information Systems (GIS) require the connectivity of adjacent two-dimensional objects. This project was undertaken as an attempt to add full 3D building models, including interiors, to the tool collection used in GIS, so the background and application of GIS, and discussions of what is meant by 'analysis' in that discipline are given first.

GIS has traditionally been an integrated discipline for a variety of two-dimensional data sets and applications. More recently '3D GIS' has started to model the third dimension – primarily by defining the exteriors of buildings in an urban environment. The data for these models has largely come from airborne or ground-based laser scanner data. From these clouds of 3D data points an appropriate building model has to be constructed. This consists of a set of adjacent flat panels, sometimes rectangular and sometimes triangular, that describe the exterior shape of the building, and sometimes its relationship with the ground surface. As a shown by Gold et al. (2006) it is possible to describe this building surface as an extrusion of the original terrain surface, using the same topological elements. This in effect treats the global Earth's surface as a single complex polyhedron, or "shell", which may be manipulated by the same process as is used in CAD systems – Euler operators – that guarantee the resulting connectivity of the complete shell subsequent to any modifications. Other approaches construct the individual shells of the building model and then simply position them appropriately on the terrain surface.

However, this approach is inappropriate if the building interiors are necessary for the particular application desired. A more complex structure is required, involving individual interior rooms as well as the adjacency relationships between them. Various approaches have been used to achieve this, in particular the non-manifold structures used in CAD systems. However, these systems have been complex and difficult to implement, as well as lacking some of the fundamental desirable features of such a structure. The main thrust of the current research has been to develop a data structure that allows for relatively simple model construction, including building interiors, which may also be integrated with the terrain to form part of a fully 3D GIS.

1.1 Objectives

The main objective of this research is to develop a data structure for three-dimensional (3D) modelling. This should be a real-time locally modifiable structure that integrates the primal and the dual graphs, along with their attributes. The main application is building interior modelling, where models will be used for path planning for disaster management. This is not easy in major public areas such as airports, shopping centres and sport venues. Recent publications by the groups involved with Lee, Zlatanova, Slingsby and Raper (Lee and Zlatanova, 2008; Slingsby and Raper, 2008) have attempted to address these issues, but the methods are still incomplete, and can be enhanced to cover a full 3D topology – for example access via ceilings and walls. But full 3D volumetric data structures are difficult to implement and do not usually address the simultaneous management of the dual structure, which is necessary for real-time query and editing.

The development of the 3D data structure with construction and navigation methods are based on previous results: the quad-edge (QE) (Guibas and Stolfi, 1985) and the augmented quad-edge (AQE) (Ledoux and Gold, 2007). The first idea is a 2D data structure for the Delaunay triangulation (DT) and the Voronoi diagram (VD) representation. The DT and VD are graphs dual to each other – one of them can be used to reconstruct the second one and other way round, but they are sometimes stored simultaneously to improve efficiency and to store semantic information of a model. These structures are also important in GIS: the DT is used for example to tessellate a terrain which is usually given as a set of points; the VD is used to answer queries about neighbourhoods and relationship between objects. The second idea (the AQE) is an extension of the QE to the third dimension. The DT triangles are 'extruded' to tetrahedra and they are linked together by the dual Voronoi diagram. The data structure to represent these cells is the QE with a little modification that allows the making of a link between these dual structures. The AQE data structure cannot be directly used in this project because of its limitations: construction operators are not fully defined – the construction process is based on

tetrahedral cells in the primal space (DT). It is assumed that a more general tool for 3D modelling will be developed: it should be possible to build the DT/VD structure as one of the possible cases.

Other ideas widely used in computer aided design (CAD) systems are also utilized: Euler operators (Braid et al., 1980), boundary representation modelling (Stroud, 2006) and the half-edge (Mäntylä, 1988). Euler operators are a set of standard operators that allows for cellular (polyhedral) construction. They directly manipulate a data structure (pointers, connections, variables, etc.). It should be possible to use these operators accompanied by the data structure in any CAD system only if the system allows for such integration. The boundary representation uses vertices, edges and faces to store information about a cell – all entities are connected and form a graph. The half-edge is used to represent edges in a model: an edge is split into two directed halves. This solution is something in between representing an edge as one element and the quad-edge which splits an edge into four quads.

To achieve these objectives the following research plan was drawn up:

- Preliminary implementation of the AQE data structure, and the development of the key navigation operations and algebra.
- The analysis of the construction operations, and the development of the atomic operations for adding or deleting primal and dual elements simultaneously.
- The construction of prototype buildings, and the validation of the construction and navigation operations in this context.
- Integrating the building structure within the embedding landscape and escape routes.
- The development of an appropriate visualization system.
- Large-scale testing of the approach.

1.2 Scope

The goal of this research is to develop a data structure that allows full 3D model construction – models should be locally and real-time modifiable; non-manifold objects should be allowed (for example two cells joined by a shared edge or vertex). Model storage is not discussed – models (geometry, topology and attributes) are stored in files on disk and all data is loaded into computer memory. A simple model for storage in a database is proposed. Managing the topology in database systems is not easy: that task needs further research (van Oosterom et al., 2002). A dual model representation that should improve the efficiency of systems based on the proposed models is more important in this research than storage space optimization. The level of detail available in a model is also not considered – the data structure is used to construct a model from a given data set and no change or reduction of details is attempted.

The developed data structure is used to model building interiors. This is only one possible application: the flexibility of the proposed solution should allow for applications in various fields, for example models that conform to the finite element method. These possibilities are not discussed in detail.

Finally the robustness and efficiency of the algorithms implemented for construction and analysis of buildings models are not discussed. It is assumed that the input data sets (geometry of a model) are valid and no further consistency checking is required. If the provided data sets are not valid (overlapping objects, missing cell faces, etc.) then errors in model construction are possible. The implementation of a data structure and the construction operators are the main focus of this research.

1.3 Publications

During this project several papers concerning the presented research were published:

- Boguslawski, P., Gold C., Ledoux, H., 2011, Modelling and analysing 3D buildings with a primal/dual data structure, ISPRS Journal of Photogrammetry and Remote Sensing, 66(2), pp. 188-197.
- Boguslawski, P., Gold, C., 2011, Rapid Modelling of Complex Building Interiors, Advances in 3D Geo-Information Sciences. Lectures Notes in Geoinformation and cartography, Kolbe, T.H., König, G. and Nagel, C. (eds.), Springer, pp. 43-56.
- Boguslawski, P., Gold, C., 2010, Euler Operators and Navigation of Multi-shell Building Models, Developments in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography, Neutens, T. and Maeyer, P. (eds.), Springer, pp. 1-16.
- Boguslawski, P., Gold, C., 2009, Construction Operators for Modelling 3D Objects and Dual Navigation Structures, 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography, Lee, J. and Zlatanova, S. (eds.), Springer, pp. 47-59.
- Boguslawski, P., Gold, C., 2009, Beyond Voronoi: The Case for Primal and Dual Data Structures, ISPRS International Workshop on Multidimensional & Mobile Data Model, Malaysia.
- Boguslawski, P., Gold, C., 2009, Unified 2D Terrain and 3D Multi-Shell Building Models, ISPRS International Workshop on Multidimensional & Mobile Data Model, Malaysia.
- Boguslawski, P., Gold, C., 2008, Construction Operators for Modelling 3D Objects, 3rd Research Student Workshop, Plassmann, P. and Roach, P. (eds.), University of Glamorgan, UK, pp. 70-74.
- Boguslawski, P., Gold, C., 2007, Atomic Operators for Construction and Manipulation of the Augmented Quad-Edge, Proceedings of the 6th International Conference on Computer

Information Systems and Industrial Management Applications, IEEE Computer Society, Poland, pp. 125-128.

- Boguslawski, P., Gold, C., 2007, Augmented Quad-Edge – 3D Data Structure for Modelling of Building Interiors, The 5th ISPRS Workshop on Dynamic and Multidimensional GIS, Jiang, J. and Zhao, R. (eds.), China, pp. 51-54.

1.4 Outline

This thesis consists of seven chapters. The first one is an introduction to the thesis and the research topic. The next three chapters describe the three disciplines the presented research is based on: GIS, mathematical preliminaries and CAD systems. They act as a literature review and help to understand some basic ideas used. The last three chapters are the main description of the project – the developed data structure, construction methods, algorithms, applications and conclusions are presented.

Chapter 1 Introduction familiarizes the reader with the research topic, the thesis structure and the objectives of the project.

Chapter 2 State of the art – GIS presents the spatial models, the differences between fields and objects, and the dimensionality of the models. Then some GIS data structures and operators are introduced.

Chapter 3 State of the art – Mathematical Preliminaries introduces basic issues concerning computational geometry. The focus is put more on problems solved in this field than on optimization methods. Graph theory is the most important field: first simple 2D graphs are described, then the duality of graphs using the DT/VD example is presented, and finally some 3D concepts are explained.

Chapter 4 State of the art – CAD introduces types, data structures and model representations used in CAD systems. Then the construction method for a cell and cell complex based on Euler operators is presented.

Chapter 5 The dual half-edge (DHE) data structure presents the main research. The data structure developed during the project is described in detail. There is also a detailed description of construction operators and modified construction versions. Simplified versions of the original

data structure are also presented. Finally the results are compared with the solutions from other researchers.

Chapter 6 Applications shows possible applications with a particular focus on building interior modelling. An example of a real building model consisted of 1300 cells (rooms, corridors, doors) is presented. Shortest paths between rooms and escape routes are analysed using the Dijkstra algorithm.

Chapter 7 Conclusions concludes the thesis. Advantages of the DHE data structure and possible future projects are presented.

2 State of the art – GIS

Early work on computer-assisted mapping – frequently referred to as automated cartography – was primarily concerned with the direct emulation of manual map drawing procedures. There was very little analytic capability in such systems, and in many cases all that they could do was to copy the drawn lines at various scales. Following from this came the capability to connect the various drawn or digitised lines to form completed regions, land areas that could be given a particular classification depending on the application.

Probably the first *geographic information system* (GIS) was developed by Tomlinson for the Canadian Goverment in 1960s (Coppock and Rhind, 1991). The project involved the production of many maps for agricultural analysis. There were several new technologies developed which were used later in GIS, for example: a scanner for rapid map digitization, a data indexing scheme, topological coding of boundaries (based on link/node idea of line encoding). In the early 1980s commercial systems were developed based on previous achievements. The most successful was probably ESRI's Arc/Info.

Burrough and McDonnell (1998) give several definitions of GIS: toolbox based definitions, database definitions and organisation based definitions. The first of these states that GIS is a set of tools for collecting, storing, retrieving, transforming and displaying spatial data from the real world. The second definition emphasises the different ways of storing and retrieving spatial data, and the third definition emphasises the role of people and institutions in their handling of spatial data. In simple terms the functionality of the GIS includes data input, data storage and management, data manipulation and analysis, and data output and display.

A map is intended as a visual representation of the real world using a predefined set of symbols, such as lines, points, polygons or text. Topographic maps depict the physical structure of the Earth's surface, such as relief, natural characteristics and man-made features. Thematic

maps display the distribution of a particular set of attributes, such as climate or population density. Traditionally maps were intended to be printed on paper, but more recently the limitations of this medium have led to computer-based graphical depiction, as attributes such as scale, level of detail and the selection of desired features may readily be modified. Even if still printed on paper occasionally, most maps are prepared from digitally stored data using a variety of computing software (Burrough and McDonnell, 1998). More recently computer visualisation has included features such as animation and 3D display. 3D display of the terrain, as in terrain models, has been available for some time, and more recently 3D depiction of building exteriors has been added.

The sources of geographic data are very varied. Primary data is measured directly from the environment using sensors: this includes photogrammetry (remote sensing images and aerial photographs) and field surveying (e.g. GPS observations), while secondary data was created for other purposes and has been modified for use within a GIS: this includes cartographic digitization from scanned and topographic maps (Li et al., 2004; Worboys and Duckham, 2004).

Spatial data may be categorised into following types: points (e.g. buildings, trees); lines (e.g. rivers, roads); regions (e.g. forests, pollution zones); networks (e.g. public transport systems represented as a graph); and partitions (e.g. land parcels represented as a set of regions together with topological relationships of adjacency or disjointedness) (Schneider, 2009).

2.1 Spatial models

The definition of space is not clear and is difficult to formulate. In different disciplines space has different descriptions (e.g. geography, mathematics or physics). A general definition to describe our world can be formulated as follows: Space is of three-dimensional extent; it has no boundaries; events occur and objects exist in space and relations between them can be specified. However on a universe scale this is not a complete description, because the curvature of space should be also considered: in modern physics Euclidean geometry does not describe space very well. Thus the relativity theory was created by Albert Einstein. In this case time is considered as the fourth dimension. Does this mean that the general definition is not valid? Everything depends on the scale. To describe the universe this may be not enough, but at a geographical scale this is appropriate. By the geographical scale a surface of the Earth, an area of a city or town, or a single building with its interior are meant.

In many cases *geometry* and *topology* are terms frequently used in GIS to represent different types of spatial information. Geometry usually refers to the positional information about an object – usually its coordinates in space. Topology refers to the spatial relationships between objects, excluding explicit coordinate information. Thus in a triangulation the data points contain the geometric information and the triangle edges express the relationships (Gold, 1987).

It is well-known that spatial relationships are not easy to determine from coordinates alone: thus the means of specifying the connectivity information for geographic data becomes a critical issue in two dimensions and even more so in three.

To analyze space with objects and concomitant phenomena it has to be simplified first. This 'compressed' version is called a model. Too many details in the final model can make investigation too complex – too little can make it inaccurate, because of the lack of necessary relations. The level of detail is dependent on the investigation purpose.

There are two main types of models: *physical* and *mathematical* (Banks et al., 2009) that can be used for various simulations. A physical model is usually made at a scale: a smaller version of a real object like a ship or plane is made because the original object is too big to analyse. A larger version is used when the object is too small for observation, for example molecules or atoms. Sometimes a full-scale model needs to be built for testing. For example a car prototype may be made to do crash tests. The method is expensive but it is justified when a big batch of a product is planned. Very often a physical model is not necessary and a mathematical model can be analysed on a computer. An object has to be digitized – described with mathematical equations and symbolic notation. This is a cost-effective method. Currently computers are more powerful and modelling methods are advanced, so a mathematical model is made even if the physical one is required – a preliminary computer analysis can improve the properties of the physical model. A further classification of the models can be done: *static* (no time flow) or *dynamic* (change over time is taken into consideration); *discrete* (the state of a model changes at discrete moments) or *continuous* (the state of a model changes continuously over time); *deterministic* (no random variables in a model; the same input values gives unique results every time simulation is performed) or *stochastic* (random sets of input variables give different results for the simulation) (Banks et al., 2009).

A slightly different emphasis focused on spatial models is presented by Wegner (2001): they can be classified with respect of their formalisation degree (scale, conceptual and mathematical models); how they deal with the indeterminism of the phenomena (deterministic and probabilistic models); or time (static and dynamic models). Other classifications are also possible (according to the resolution, extent in space, time, attributes, etc.).

In the case of GIS models it is important to include relationships between the geographic environment and its representation in the model (Worboys and Duckham, 2004). For example, connections between rooms in a building are important for radio wave propagation (Becker et al., 2009) or fire spreading simulations; connections in a city road network are important for simulation of traffic.

2.2 Fields and objects

While many concepts of space have been proposed, the most widely accepted distinction is between *objects* and *fields* (Couclelis, 1992; Worboys and Duckham, 2004). The difference between them can be simplified to the difference between a raster and vector representation. It appears that people perceive a world as fields as well as objects, depending on scale and purpose. Alternatively a world was considered as empty space with entities distributed in this space – space can have a certain property only in place taken by an entity; in other 'empty' places space does not have any properties.

Objects are discrete entities with no expressed relationships between them in the map: in practice they are composed from the three primitive elements of points, lines and polygons. Such representation means that these entities are only approximations of real objects. Some approximation is assumed in modelling – this is acceptable. But there is another problem: how to outline the boundaries of an object? Some boundaries exist only in the human imagination (or are the result of human cognition) they are called *fiat boundaries* (e.g. geopolitical boundaries between countries). Physical discontinuities existing in the world without human cognition are called *bona fide boundaries* (e.g. geographic boundaries: coastlines, rivers, walls, etc.) (Smith, 1995). Nevertheless boundaries between objects in the real world are often fuzzy (Couclelis, 1992) – for example, where a boundary between a valley and a hill is situated: it is not possible to draw a thin line dividing these two formations without uncertainty.

Objects must have described attributes and location information in order to have meaning. In many cases no spatial relationships are defined between objects in a map, but in some cases the entities may be related to a *field* model of the space: either they are point samples of the field, or inversely the objects are extended to fill the overall field (Voronoi diagrams for example (Gold, 1987)). Fields on the other hand represents continuous phenomena: as these cannot directly be represented in a digital computer they are usually converted into tessellations, subdivisions of the plane. These tessellations may be regular or irregular: grids are a typical example of regular tessellations, and triangulations of the irregular form. A choropleth map of an attribute, with each polygon containing just one class of the attribute range, is a common form of tessellation. A triangulation (TIN) with each vertex associated with an elevation value is perhaps the most frequent discretization of a continuous surface.

Methods used in computational geometry and CAD systems are applied in this research. They are designed for the geometric representation of 'objects' which uses points, lines, polygons and polyhedra to construct them. Therefore the focus is put more on objects than fields.

2.3 Dimensionality

Despite the previously described concepts of a space representation dimensionality is a very important factor in modelling. There are different problems to solve in the case of simple two-dimensional and much more complex three-dimensional representations. The 3D case is more difficult to implement and complex geometric algorithms are required. Thus to find a compromise between simplicity and functionality indirect 'dimensions' are considered (e.g. 2.5D, 2.75D, etc.). These are 3D models simplifications that allow for more efficient modelling and analysis. However the development of faster computers results in a growing interest in full 3D models. They are more resource demanding (computer memory, storage space, and processor time) but the results are more precise or even not possible with models of lower dimension. A brief review of dimensionality concepts was presented by Penninga (2008):

2D

Traditionally, systems were two-dimensional and represented the world as thematic maps (see Figure 2.1a) – only points, lines and polygons are used in 2D space.

2.5D

A location in space is represented with two coordinates x and y; z – the height is only an attribute that is used for example for visualization. The 2D object is embedded in 3D space (see Figure 2.1b) – the structure is two-dimensional, and the third dimension is represented by attributes (Tse and Gold, 2004). 2.5D modelling is used in terrain modelling – the terrain is represented as a surface (topologically identical with a surface of a sphere) where each location can be defined with only two coordinates. Vertical faces and more complex structures like bridges, tunnels or building interiors cannot be represented.

2.5D+

This is an extension of the previous concept, but vertical faces are also allowed (see Figure 2.1c). This is used in "block-shaped" models of building exteriors where roofs are horizontal planes with a height assigned; walls represented as vertical planes are reconstructed by extrusion (Ledoux and Meijers, 2010).

2.75D

To permit more advanced construction (e.g. bridges, tunnels, etc.) the z coordinate is required. Tse and Gold (2004) presented how to model the surface of the world with tunnels and bridges using TIN and CAD functions (e.g. Euler operators) (see Figure 2.1d). However this method does not allow for the construction of 3D objects with separate volumes. From a mathematical point of view these models are 2-manifold shells (polyhedra) with holes.

3D

Generally 3D data models can be divided into four groups: 3D geometric models, 3D topological and graph models, 3D city models and 3D CAD models (Lee and Zlatanova, 2008). 3D object modelling and analysis is relatively new in GIS (Zlatanova, 2000a). There is no easy way to transform existing 2D data models into a 3D version (Thomsen et al., 2008); therefore there are several spatial models used in accordance with a particular application, for example the constructive solid geometry or boundary representation.

One of the approaches in 3D modelling is the construction of separate volumes connected together into one complex (see Figure 2.1e) – each volume is bounded by faces, edges, and points (the boundary representation is described in Section 4.2); thus in a 3D model there are 3D volumes, 2D faces, 1D edges, and 0D points available, where a 3D volume may be bounded by a 2D map (this is 2-manifold) (Gold, 2006). Construction of models represented with a set of polyhedra is allowed in CAD systems, but unfortunately, they are not topologically connected; such connections are calculated from geometry each time they are required (Lee and Zlatanova, 2008), which is not efficient.

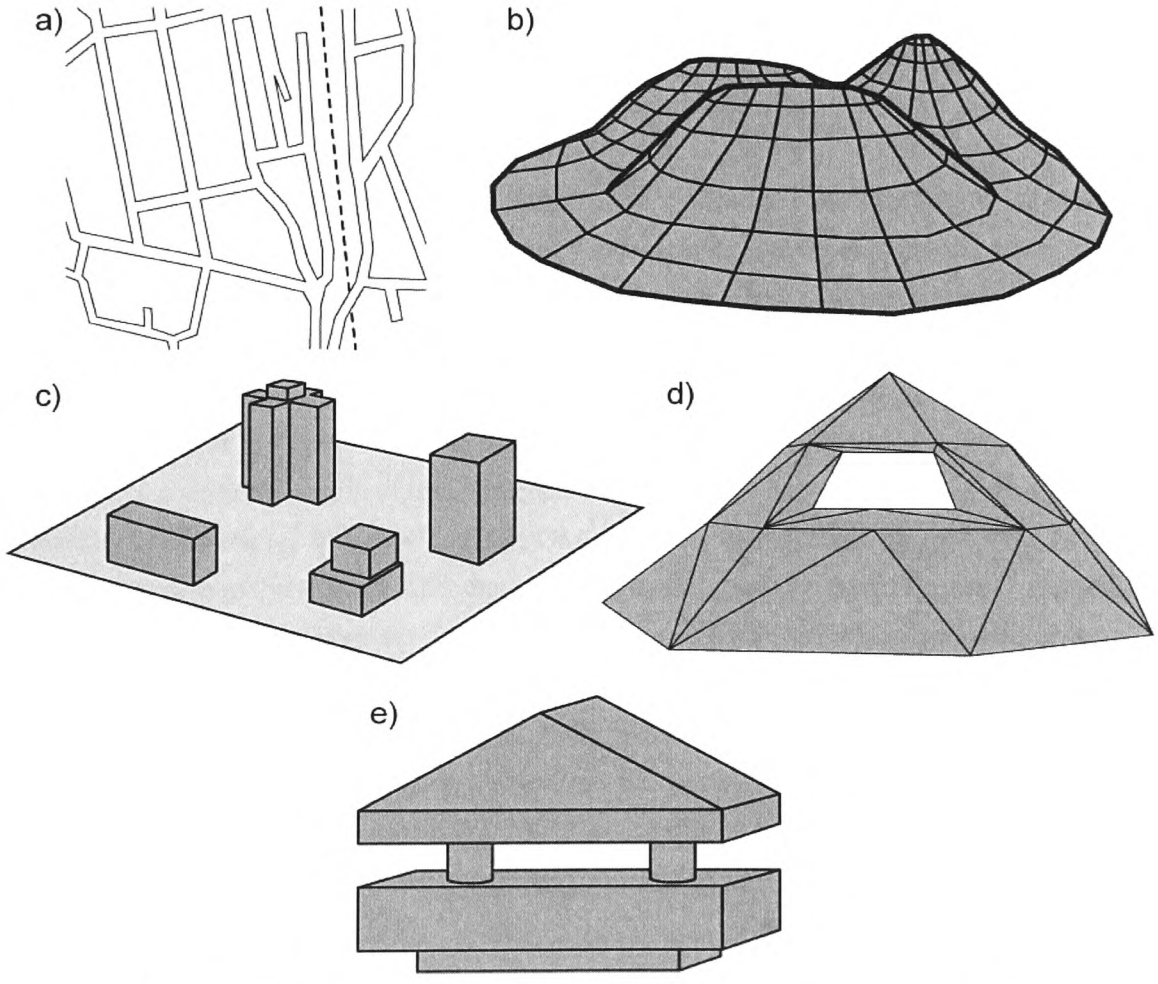

Figure 2.1 Model dimensionality: a) 2D; b) 2.5D; c) 2.5D+; d) 2.75D; e) 3D.

There are applications for which the spatial connectivity between objects is crucial (e.g. terrain modelling, tessellations, networks): simulations and advanced analytical tools demand a full topological connectivity for more efficient analysis (Gold, 2006). Nevertheless using CAD modelling methods in GIS systems is an interesting idea. Information could be assigned to the objects representing real-world objects and they could be identified in a 3D environment; and advanced functionality would be possible (Stoter and Zlatanova, 2003).

2.4 3D GIS

3D geographic information systems (3D GIS), which should include a full 3D model with integrated 3D geometry, 3D spatial relations (topology), and semantic information in order to analyze and visualize this model, are strongly needed (Coors, 2003). Currently research is divided into 3D City GIS and 3D Geological Modelling (Yanbing et al., 2007).

Recently much research has been done in the 3D city modelling field (Kolbe et al., 2008), which is important from the 3D GIS point of view. Virtual city models are currently of interest for many applications like disaster, cadastre, surveying, urban planning and land management systems. There are several possible representations, for example the Urban Data Model proposed by Coors (2003) is a topological data model for 3D GIS that was applied to build a city model of Darmstadt; the data captured by a telecommunications company was used to simulate a mobile network.

Another 3D standard is the CityGML information model and exchange format: large areas of a city or region can be described and stored. Several cities in Germany provide 3D city models using CityGML (Kolbe, 2009). Buildings, terrain models, city furniture, vegetation, land use, water bodies, transportation (e.g. streets, railways) are defined in thematic modules which can be extended in the future. The data model in CityGML is based on the GML3 standard which permits one to define the spatial properties of a model – mainly geometry, but also the topology of a model may be included: concepts for attributes, relations, and generalization hierarchy definitions are also provided. CityGML supports five levels of detail (LOD): LOD0 is the coarsest, essentially this is a 2.5D digital terrain model; LOD1 is a block model – buildings are represented as blocks with flat roofs; in LOD2 more complex buildings can be modelled – complex roofs, installations like stairs and balconies are available; LOD3 allows for architectural models – detailed walls, roofs, doors windows, etc. are possible; LOD4 completes LOD3 and includes interior structures like rooms, doors, stair, furniture, etc. It is possible to represent the same object simultaneously in different LODs. This is also important in disaster or emergency planning and simulations – not only the resolution, precise geometry and topology of a model are important but also the spatial dimensionality of the data: a 2D representation is sufficient for terrain modelling in most cases but not in the case of city objects like buildings. A

3D representation is essential in the planning of escape routes from buildings, indoor navigation, flooding scenarios simulations, etc. The main focus of this research is put on building interior modelling and the relationships between rooms thus only CityGML models with LOD4 are taken into consideration.

CityGML provides several classes. The root class is *CityObject* – an abstract class which inherits from the GML *Feature* class. Except for *name*, *description* and *id* there are *creationDate* and *terminationDate* attributes which allow the representation of an object change over time. *CityObject* may also be associated with objects in different datasets or databases using *ExternalReferences* – this can be used, for example, to link the model with the original source from which the model was derived. A diagram representing the top level of the CityGML hierarchy is shown in Figure 2.2.

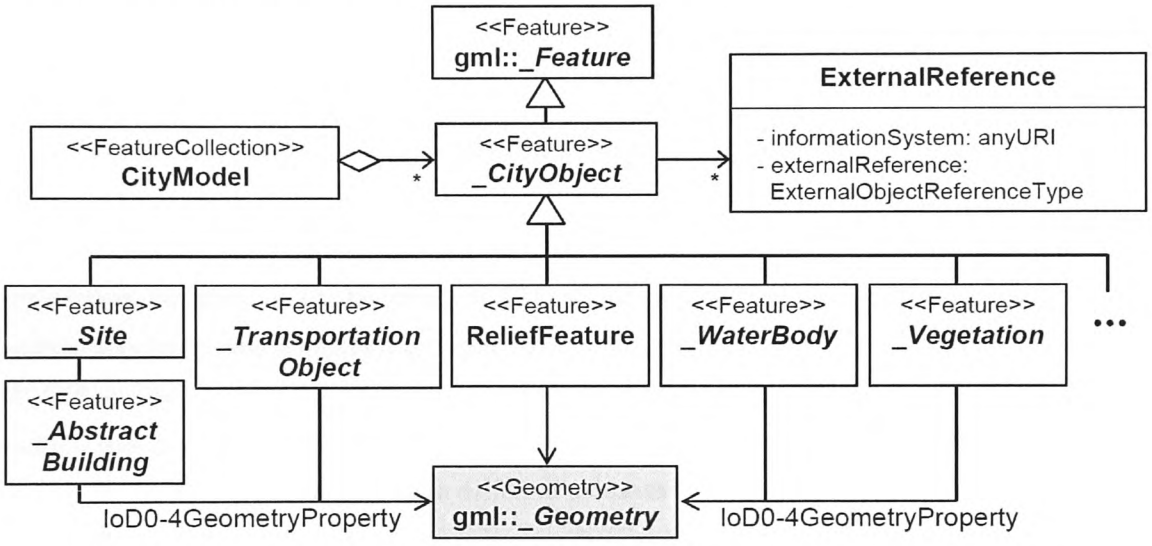

Figure 2.2 UML diagram of the top level CityGML class hierarchy. (Kolbe, 2009)

The base class for building modelling is *AbstractBuilding*. *Building* and *BuildingPart* are non-abstract classes derived from the *AbstractBuilding*. *Building* may contain *BuildingParts*. Because they are derived from the *AbstractBuilding* class a recursive aggregation may be implemented (Kolbe, 2009). *Building* and *BuildingParts* inherit several attributes from *AbstractBuilding* which describe properties of a building. Also surfaces of each part of a building can be included and their geometry defined in GML.

Another standard for 3D building and terrain modelling worth mentioning is the Industry Foundation Classes (IFC) Standard (ISO/PAS 16739, 2005). Unfortunately streets, vegetation objects, water bodies, and others are not included thus this is not suitable for a complex city modelling. However a lot of research is done to translate between IFC and CityGML, for example (Isikdag and Zlatanova, 2009); IFC data sets are important sources for building models including LOD4 (Kolbe et al., 2008).

The geometry of an IFC building model is represented using Constructive Solid Geometry (CSG), Sweeping, Boudary Representation (B-Rep), or a combination of these methods. Building elements can have multiple representations (e.g. CSG and B-Rep) (Isikdag and Zlatanova, 2009). A *building* is composed of *building parts*; a *building part* is described by attributes (e.g. function, usage, construction year) and aggregates *variants*; a *variant* contains geometric and semantic model of the *building part*; a *variant* may describe a specific level of detail or different planning states of the building (Benner et al., 2005).

There are some similarities and some differences between IFC and CityGML – the CityGML LOD modelling is less flexible and general than the concept of variants (Benner et al., 2005).

Regardless of the standard or data format used for model representation it is necessary to ensure the consistency of the model before further analysis. Achieving consistency for 3D city models and the derivation of indoor models for route planning were presented by Gröger and Plümer (2010a; 2010b).

Geology is another field which benefits from using 3D GIS. 2D models do not allow for analysis of all available spatial exploration data – only a 3D model can explicitly represent the 3D earth (McGaughey, 2006). Usually geological models, used for example in mineral exploration, represent an ore boundary as a 3D wireframe mesh with estimated grade values, but without topology, which makes the model difficult to edit and analyse; other conventional CAD, 2D GIS and grid based systems are also inadequate (McGaughey, 2006). Thus a new modelling method was required. The *common earth model* used in the oil and gas industry integrates spatial exploration data collected with different methods into one unified model (De Kemp, 2007; McGaughey, 2006). This solution can answer geological and geophysical queries, and can be shared by specialists from different fields engaged in exploration. The common earth model can be implemented as a regular partitioned voxel model with properties mapped to internal vertices – however, depending on requirements, cells can have different regularity and size (De Kemp, 2007). It is important to understand that geological structure modelling is different from CAD modelling where surfaces are constructed from a given point set. Typically it starts with a data set obtained from several drillholes (McGaughey, 2006) and geological surfaces are estimated using geometric modelling algorithms (Mallet, 2002). Geocad is a software example that is developed using methods necessary to achieve the common earth model (McGaughey, 2006).

2.5 Data collection

To build a GIS model first geospatial data has to be collected. Geographic (or spatial) data could be typified as any information whose value would be lost when its precise location was not

specified. Input sources to get raw data include: remote sensing (airborne laser-scan, aerial photographs, photogrammetry), ground-based observations (surveying, GPS), census figures and statistics. They are used to sample the natural terrain as well as man-made objects (e.g. buildings – roofs and interiors, roads, etc.). The collected data needs to be processed (e.g. noise reduction, preliminary object recognition) and then is used to reconstruct models that can be analysed and simulations can be performed. Four general approaches are considered for automatic 3D model construction: however optimal reconstruction is supported by manual and semi-manual methods (Stoter and Zlatanova, 2003):

- bottom-up – footprints from existing 2D maps are extruded: the height of objects is obtained from laser-scan data, surveying, GPS or photogrammetry data; buildings are represented as simple blocks. Topologically consistent building extrusion from their footprints is presented by Ledoux and Meijers (2010).
- top-down – based on roof information obtained from aerial photographs or airborne laser-scan data; accuracy depends on the source data resolution. An important remote sensing tool is airborne laser surveying (also called LIDAR) (Li et al., 2004). Research on automatic 3D city modelling from LIDAR is presented by Tse et al. (2008).
- detailed reconstruction – 3D point cloud data from laser scan or 3D shapes obtained from aerial photographs are used for fully automatic object reconstruction: a disadvantage is the high complexity of algorithms and time consumption.
- combined approach – different data sets are combined to obtain all the details of the modelled area (e.g.: laser data, topographic data, aerial photographs, maps, etc.).

2.6 Data representation and visualization

In GIS the modelled area of interest is usually large with many objects included, for example the area of a city with terrain, buildings, roads, etc. The amount of data representing such a model is huge, especially as adding the third dimension to a model increases the amount of data significantly (there are more geometric data and spatial relations). Widely used relational databases can easily handle separate (discrete) objects which can have their unique IDs and attributes. But links between objects are still a challenge to resolve (perhaps object-relational databases are able to easily handle this) (Gold, 2006). It is not a problem to store topological references, but the problem is how to efficiently implement topological models and complex topological queries in a relational database (van Oosterom et al., 2002).

Photo realistic images of complex models require a lot of computation and are used to visualize the final model (simpler techniques like shaded images, wireframe or surface representation are used during the design process). Polyhedral faces are usually approximated by triangles and curved surfaces decomposed into triangle meshes before they are visualized –

graphic libraries and hardware are optimized to process and draw simple shapes very fast (Rossignac and Requicha, 1999).

Additional problems are present when 3D models (e.g. 3D-GIS city models) are managed using web-based interfaces: visualization of a model requires a huge amount of data to be sent, which results in a long transfer time. A visualization technique based on flexible levels of detail management was presented by Coors and Flick (1998): first objects at a low level of detail are sent from a server to a client to display them as soon as possible, then more details are transmitted and visualized progressively. To make the transmission faster data sent over the Internet is compressed: Coors and Rossignac (2004) proposed an efficient algorithm of triangle mesh compression (triangle meshes are used for visualization and exchange of 3D models).

Visualization of geospatial data received from servers over the Internet can be realized through a 3D geospatial browser, for example Google Earth or Bing Maps Platform (previously Microsoft Virtual Earth) – currently Google Earth is the most popular (Isikdag and Zlatanova, 2010). In such browsers a virtual globe representing the Earth is covered with a layer of satellite images. Also aerial images and map data are available. All photographs and other data are geo-referenced thus a location can be browsed by entering coordinates. The image resolution in big cities in the US, the UK, and Western Europe is high – 15cm to 1m details are recognizable (Stefanakis and Patroumpas, 2008). 3D terrain data is available for most places in the world thus they can be visualized in 3D. Also users can add their own data like points of interest and geometrical objects with associated semantic information. For example building models can be added by a user and visualized in the surrounding of an existing area. Buildings and other objects are represented as boundary representation models that can be associated with textures (e.g. a texture of a building facade). They are stored in KML files which is an XML based format. To save storage space a simple building can be represented as a footprint (i.e. a floor plan defined by a polygon) which is extruded to the given height. This method also reduces the time needed to load a large data set, for example representing the whole city. But this method is not adequate to represent complex buildings. In the second method all faces of a building model are stored as a set of polygons representing the geometry. Currently hundreds of 3D city models are available in Google Earth. In this way municipalities promote their cities as a tourist destination and business location. It is also used as an urban planning tool to show how new buildings would impact the existing environment (Isikdag and Zlatanova, 2010).

2.7 Data structures

To analyse the world with its objects and phenomena using a computer it is necessary to create a digital model. An appropriate data structure chosen for modelling depends on the application and dimensionality of the model (Gold, 2005).

A *triangular irregular network* (TIN) is very common in GIS; this is used for example in a digital terrain surface modelling (Li et al., 2004). Construction of a TIN for irregularly distributed points is not particularly easy.

The Delaunay triangulation is widely used in GIS. In this triangulation non-overlapping triangles fill the area without holes or breaks; a circle circumscribing each triangle does not include any other points from the input set. There are many construction algorithms for this kind of triangulation; they are divided into static and dynamic; the static means a fixed set of points that is not changed later; the dynamic triangulation allows for adding new points to a network, but every time a new point is added the network is changed to fulfil the 'circumcircle condition'.

One of the most important factors that affect the efficiency of computation is the data structure used for representation of the spatial model. Different data structures can be used to represent the same spatial data model (Ledoux, 2006). For example triangular meshes in 2D or tetrahedral meshes in 3D can be represented with half-edges, radial-edges or other data structures. Tse and Gold (2004) combine the quad-edge data structure and the boundary representation to implement the TIN. Their idea allowed them to extend the TIN, used to represent 2D and 2.5D models, to 2.75D – not only vertical faces are allowed in their models, but also holes and caves. They also introduce a set of Euler operators for TIN construction that conforms to CAD systems.

Another data model related to the current research is the *3D Navigable Data Model* (3D NDM) (Lee, 2007) developed to model microspatial build environments (e.g. multilevel buildings, see Figure 2.3a). This is based on two components: a logical and a geometric data model; and permits one to represent adjacency, connectivity and hierarchical relationships between 3D objects. As a starting point architectural plans stored in CAD files or scanned blueprints can be used to create the 3D geometric model – a wall structure is extracted using a vector-based medial axis transformation in the case of CAD files and a raster-based thinning in the case of blueprints (Choi and Lee, 2009). A graph of connections between rooms – the logical network – is derived using the Poincaré duality (see Figure 2.3b): a room is represented as a dual node; a wall between adjacent rooms – as an edge connecting dual nodes (Lee and Kwan, 2005). To make 3D spatial queries (e.g. a path searching algorithm) possible to implement the geometric network model is required (see Figure 2.3c) (Lee, 2004). All corridors represented as nodes in the logical network (nodes n_6 and n_{12} in Figure 2.3b) are converted into linear features (edges e_1 and e_2 in Figure 2.3c) using the straight medial axis transformation. Then nodes representing rooms originally connected with the corridors are projected and linked to the newly created edges. 3D spatial query performance can be improved using a hierarchical network structure used to represent a transportation network (Lee and Kwan, 2005) – the first

(highest) level is formed by nodes and edges representing vertical connections between floors, for example stairways represented by dual nodes are connected by vertical edges; the second level describes connectivity on a particular floor and is formed by nodes and edges representing a corridor and adjacent rooms; the third level describes connectivity in a building subunit, for example rooms which are not directly connected to the corridor but through the room adjacent to the corridor. The highest level of this network can be compared to the main motorways connecting cities, and the lower level links represent local roads connecting towns and villages with motorways.

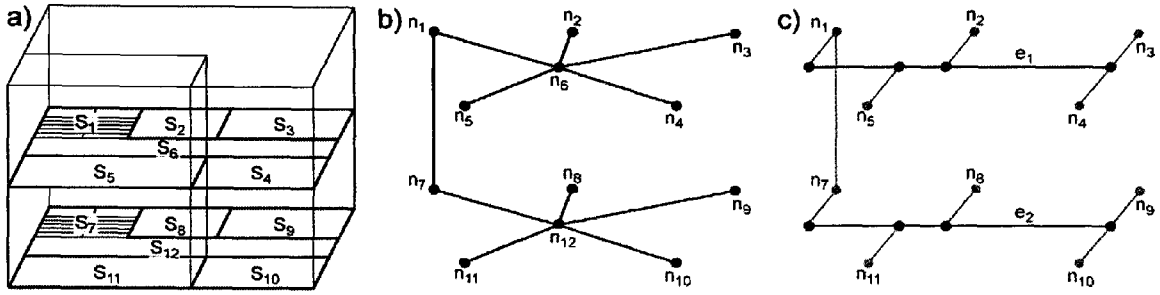

Figure 2.3 3D Navigable Data Model: a) 3D spatial objects; S_1-S_{12} – rooms; b) logical network; n_1-n_{12} – nodes representing original rooms; c) geometric network e_1, e_2 – corridors transformed into linear features. (Lee and Kwan, 2005)

Elements required for the geometric network are nodes and edges; thus the base classes of the model are as follows: Node, Edge, and Network. The Node class consists of *id* and *x*, *y*, and *z* coordinates; Edge –*id*, *start node*, and *end node*; Network – *id*, *list of all nodes*, and *list of all edges* in the network. Attributes can be attached to nodes (e.g. room use, occupancy data, data collected from temperature, smoke gas, and fire detectors) and edges (e.g. occupant movement, corridor capacity, traffic flow impedance, etc.). Other subclasses are also present in the 3D NDM and are used to define a pedestrian transportation system (Lee, 2007). The geometric network can be used in emergency response systems for path finding, allocation, and tracking analyses (Lee and Zlatanova, 2008).

Other 3D data structures and models that include topological information between objects, which are important in GIS systems, are listed and compared by Zlatanova et al. (2004): the Formal Data Structure (Molenaar, 1992), the Tetrahedral Network (Pilouk, 1996), the Simplified Spatial Model (Zlatanova, 2000a), and the Urban Data Model (Coors, 2003).

A solution based on G-Maps and cell-tuple structures to handle the topology of 3D models in GIS is presented by Thomsen et al. (2008); a topological model of the Osnabrück Palace was integrated into a database management system and combined with a 2D city map. The same idea of G-Maps is proposed for geomodelling applications by Mallet (2002).

2.8 GIS operators

Spatial data can be handled in many systems (e.g. CAD, database management systems, etc.). However to use them in GIS systems they must provide spatial analysis (Goodchild, 1987). They should answer queries about the geometrical, topological and statistical properties of the spatial data. The result is based on the objects' locations. Albrecht (1996) introduced a set of 2D GIS operators which uses attribute and location information attached to objects. A new set of 3D GIS operators based on Albrecht's operators were proposed by Ledoux (2006):

1. Search: interpolation, (re)classification, thematic and spatial search;
2. Location Analysis: buffer zone, overlay, and creation of the 3D Voronoi Diagram;
3. Visualization: isosurface, slicing;
4. Distribution/Neighbourhood: cost, proximity, and nearest neighbour;
5. Spatial Analysis: statistical and pattern analysis, shape;
6. Measurements: distance, area, volume and centroid.

However in this thesis a narrower set of operators concerning topological relations between objects in 3D space is more important. They are defined in the 9-intersection model (Egenhofer, 1995) and systematized by Zlatanova (2000b). These relations are important in GIS and answer queries about spatial relationships like neighbourhood, inclusion, coverage, etc. In this research the 'meet' relationship is of great significance, because all elements in the described models do not overlap and are linked into one complex without 'empty space' (space is tessellated), therefore each element is adjacent to others from its neighbourhood (they 'meet'). Thus a dual graph can be used to represent topological relationships between elements as this graph interprets the 'meet' relation between 3D objects (Lee and Zlatanova, 2008).

2.9 GIS and disaster management

In previous sections some general issues concerning GIS background, space dimensionality, and data structures were described. Disaster management is another topic connected with this research. This is also a part of GIS science as geo-information is crucial in disaster management especially when building interiors are considered (Pu and Zlatanova, 2005).

Recent events like terrorist attacks in New York in 2001, London in 2005, the Indian Ocean tsunami in 2004, or the Katrina hurricane in 2005 that affected urban areas (e.g. buildings, streets, etc.) showed that a fast response and real-time disaster management systems are vital and can save lives (Goodchild, 2006). This is especially important when people are trapped in a multi-level building where the disaster environment is changing very fast (Kwan and Lee, 2005). In such a micro-spatial environment 3D GIS data models improve the emergency response speed. Because a short period of time may introduce significant changes the real-time systems supported by 3D GIS data models connected with transportation systems would be

more efficient and thus may reduce casualties. Another important factor that improves emergency response plans is information obtained from sensors in a building (e.g. fire, smoke, occupancy pattern, or the number of people in each room). Pu and Zlatanova (2005) also emphasize the importance of models that include both geometry and logical connections inside a building – this combination makes them powerful for evacuation routes planning. The logical model is used for shortest path computation while the geometry is used for model and escape route visualization. The logical model representing topological relations between rooms may be represented as a network (Lee, 2007), thus for the computation of the shortest path between a room and the closest exit the Dijkstra algorithm (Dijkstra, 1959) can be used. Lee and Zlatanova (2008) proposed a 3D data model to represent multi-level built environments for emergency response applications; this is a combination of three models: the 3D geometric model, the 3D topological model, and the 3D city model used for visualization.

3D model of a building is only a static representation. It can be further analysed and used in simulations. Choi and Lee (2009) presented a building evacuation simulation. A building interior is represented with the 3D geometric network which is a 3D geometrical and topological model. This model is integrated with an agent-based model. People inside the building, walls and rooms are considered as agents, but only people are moving agents. Human behaviour is described by equations taking the crowd dynamics into consideration – people may move faster than usually, they may push each other, they may cause jams at exits, etc. It is assumed that people know where the closest exit is located. The evacuation process is simulated through time. Properties like the number of moving people and the average speed are analysed. Such simulations can help to plan new buildings and answer question about the number of doors in rooms or emergency exits – it may be crucial in public buildings like cinemas, hospitals, shopping malls, etc. Also the maximum number of people allowed in a room can be recommended for safety reasons, for example in offices.

Another emergency management system for buildings proposed by Filippoupolitis et al. (2009) is a real time, agent based system which can be integrated with a sensor network. It can be used not only to simulate a building evacuation but also to simulate rescuer and mobile robot exploration of a building. Information from sensors, like fire spreading, affects the model (e.g. blocked exits) and agents (e.g. state of health). Evacuation routes are computed in real time taking sensor readings into consideration. The best escape route may be indicated on panels placed throughout the building or by wireless communication devices. These devices may be carried by rescuers or robots which also can help to communicate with trapped people. In the presented application a building interior is modelled as a graph with special nodes (e.g. entrances and staircases). The building model is divided into sections controlled by separate

simulators and connected by 'bridges', for example each floor in a building is represented as a graph with a 'bridge' connection between floors.

3 State of the art – Mathematical Preliminaries

In this chapter some references to computational geometry can be found, but this is a wide field focused on optimal algorithms. In this thesis solving problems in this field is of greater importance than optimization methods.

Computational geometry is a research discipline started in the late 1970s (Berg et al., 2008). Its growth is connected with a need for optimal algorithm design and analysis in fields like computer graphics, robotics, GIS, CAD, molecular modelling etc. Geometric problems can be solved in many ways, but very often existing algorithms are slow or difficult to understand. However recently developed algorithmic techniques improve and simplify previous approaches and algorithms are more efficient and simpler to understand and implement (Berg et al., 2008).

The development of a geometric algorithm is a three-phase process (Berg et al., 2008). First a problem is simplified – all special or degenerate cases are ignored. It is much easier to solve the problem without nasty cases that introduce a lot of complexity. When a simple solution is found then the algorithm is redeveloped to handle cases ignored in the first step. However adding instructions that manage degenerate cases one-by-one is not a solution: the algorithm should be as flexible as possible and integrate all these cases into the one general case. The last phase is the implementation, where robustness is a difficult problem to solve. There are methods that provide exact arithmetic: very often available as external libraries that can be added to a program. They are slow and not always fit for applications where computation speed is essential. Sometimes it is better to use a less robust method and detect inconsistencies to avoid program errors. There is no one good way of dealing with precision in a program – a chosen method depends on the application (Berg et al., 2008).

There are different geometrical problems in different disciplines (Berg et al., 2008):

Computer graphics is concerned with displaying models stored in computer memory. Two-dimensional graphics is not sophisticated and generally involve the intersection of primitives (e.g. lines intersection, polygons intersection, the subset of primitives lying within a polygon, etc.). Three-dimensional geometry queries are more complex. Applications like 3D games simulating a real world are very demanding. Scenes are built with many objects. Each object (a boundary of an object) is usually represented as a triangular mesh. Thus a scene can contain several million triangles. The more realistic the effect the higher the density of meshes required and the higher the number of triangles in the scene. To display a realistic picture of the scene on a computer screen the light, textures, and other elements need to be taken into consideration. Algorithms for fast hidden surface removal, collision detection or shadow computation are crucial in displaying 3D scenes. Computer games are not the only example – 3D simulations of real phenomena are also very demanding (e.g. fluid or gas mechanics). In this case the robustness of the analysis is more important than the speed of visualization. However both cases are concerned with geometric problem optimization.

Geographic information systems (GIS) store data as thematic layers/maps: road or railways networks, electricity lines, rivers, landforms, etc. Each layer contains information about one feature. However relations between layers are essential when a particular region is analysed, e.g. what is the forest area in South Wales; are there any pipes, electrical lines, or rivers crossing a designed motorway; how many mobile base stations are located within a 10 km radius of Cardiff? All these questions can be answered by combining two or more maps and looking for intersections between regions of interest. This does not appear to be a complex problem, but geographical data sets are usually large. Thus efficient algorithms for seeking out the required data are crucial. Another kind of problem is interpolation: the height of the terrain is usually given for some sample points, and an approximation at other positions is necessary for analysis. The question is: what sample points should be taken for interpolation, and what is the best method of the approximation?

Computer aided design (CAD) helps with product design using computers. Contemporary models are three-dimensional and detailed. Machine parts, buildings, house decor, circuit boards and many other items are modelled by computer and tested before they are created in the real world. Packages used in CAD solve many problems: intersections and unions of objects, decomposing objects into simpler shapes, fast and realistic visualizations. The precision is crucial; the final model can be manufactured in reality.

There are more disciplines (e.g. robotics, molecular modelling) that deal with geometric analysis. New algorithms and optimal data structures are necessary to solve problems faster and more precisely, especially as models and data sets get bigger and a fast (often real-time) analysis is required. Many standard computational geometry problems are solved by using the CGAL

library (The CGAL Project, 2010) that may be included in a computer program. This contains data structures, geometric primitives, algebra, efficient and reliable algorithms, etc. thus a time consuming implementation of well known algorithms is not necessary. CGAL is not used in this project because dynamic primal and dual data structures are not addressed in their work, or work which is a particular concern of the current research. However, CGAL would be appropriate for testing polyhedron overlap and similar problems.

3.1 Graphs

There are many problems that can be solved using a network. For example, travel time from A to B, link throughput, etc. can be calculated for a system of roads, railways or electric lines. These systems form networks which can be represented with a formal model – a *graph*. This can be described as an abstract representation of a group of objects connected in pairs by links. It can be used for simple modelling of spatial relationships (i.e. connectedness) between elements of space – the topology of space.

The precursor of graph theory (the mathematical study of graph properties) was Leonard Euler who tried to solve the Königsberg bridge problem in 1736 which is an example frequently utilized in the literature (Penninga, 2008; Weisstein, 2005; Worboys and Duckham, 2004) and illustrating the use of graphs to solve a real life problem. There were seven bridges in the city connecting two banks of a river with two islands as presented in Figure 3.1a. The question was if it is possible to traverse all the bridges only once and come back to the point where the trip was started. The answer was negative – it is not possible to get across all bridges exactly once. It should be noted that the Königsberg bridge problem is not a simple graph example as there are multiplied edges between two nodes – duplicated edges connecting the same nodes are allowed in a multigraph. Mathematically the problem comes down to looking for a Eulerian cycle in a multigraph with four nodes and seven edges (Figure 3.1b).

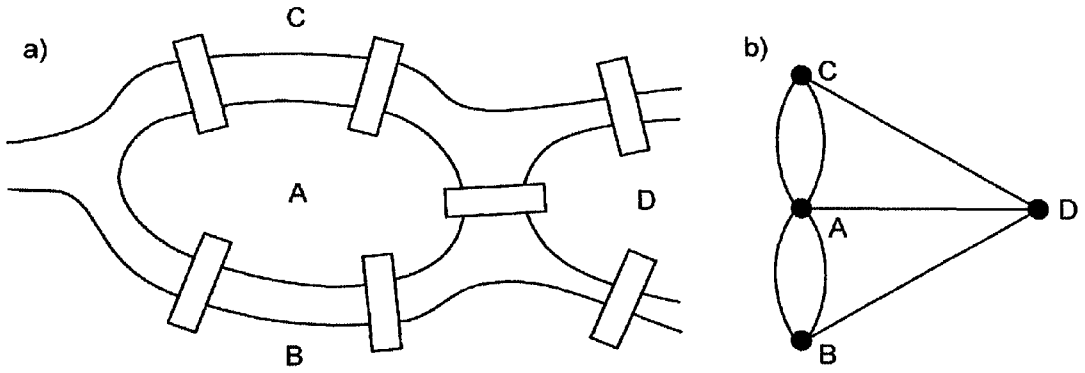

Figure 3.1 The Königsberg bridge problem: a) seven bridges of Königsberg; b) graph representation.

A *graph G* (Figure 3.2a) is defined as a finite and non-empty set of vertices V and a set of unordered pairs of distinct nodes (edges) E: $G=(V, E)$; vertices x and y of graph G are incident with an edge e when e joins x to y: $e = xy$ (Worboys and Duckham, 2004). Vertices are also called nodes or points; edges may be called arcs or lines. In this thesis if it is not explicitly stated the graphs are considered to be *connected* – all vertices in a graph are connected by edges.

Some extensions of the abstract graph are useful: directed and labelled graphs (Worboys and Duckham, 2004) are two examples. The direction of edges is distinguished in a *directed graph*; this may be useful in the modelling of a street network with one-way streets in city centres. In a *labelled graph* labels (numbers or strings) may be assigned to each edge – labels are used to assign attributes describing properties of a link; for example, this allows the storing of information about distances, travel time or traffic in a road network. It is possible to present additional graph variations (see Figure 3.2) based on these two examples. More than one edge between two vertices is allowed in a multigraph (Figure 3.2b). Multiple edges and graph loops are allowed in a pseudograph (Figure 3.2c); a graph loop is an edge with the same vertex at both ends – a vertex is self-connected. There are two versions of labelled graphs: the first one is as described above – labels can be assigned to edges (Figure 3.2d), while in the second version labels can be assigned to vertices (Figure 3.2e). In a simple graph edges are undirected, but if a directedness is introduced to edges the graph became a directed graph (Figure 3.2f), or an oriented graph (Figure 3.2g) if all edges have a unique direction (are not bidirected). If two types of graphs (directed and labelled) are combined the resulting graph is called a network (Figure 3.2h).

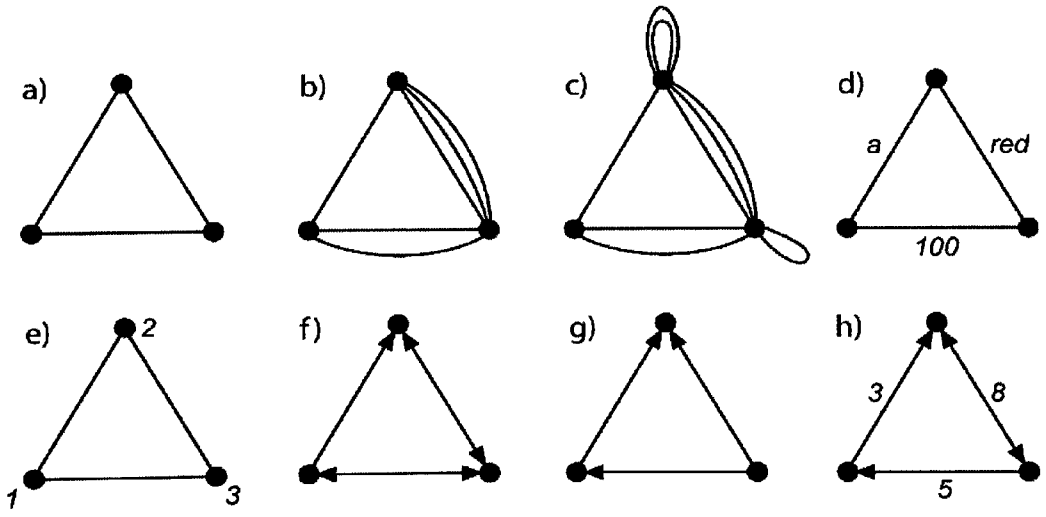

Figure 3.2 Graphs: a) simple graph; b) multigraph; c) pseudograph; d) labelled graph (edges); e) labelled graph (vertices); f) directed graph; g) oriented graph; h) network. (Weisstein, 1999)

To analyse relations between objects using graphs classified in the previous paragraph some basic terms should be introduced. A *path* between two vertices in a graph can be represented as a sequence of connected edges and is denoted by vertices that are visited. In a *connected graph* a path can be determined for any two vertices (if a path exists for any two vertices in a graph the graph is a connected graph). When the first and the last vertex of a path is the same vertex and the path contain at least one edge is called a *cycle*. A cycle that visits all the vertices exactly once is a *Hamiltonian cycle*, while a cycle that visits all the edges exactly once is an *Eulerian cycle*. For example, if all edges of a graph can be drawn on a piece of paper without a break (in one go) there is an Eulerian cycle in the graph. A special class of graphs without cycles is a *tree* (see Figure 3.3a). There is one distinguished vertex called a *root*; vertices connected to the root are its descendants, and they can also have their descendants and so on; a vertex without descendants is called a *leaf* (see Figure 3.3b). A tree is often used as a data structure in computer science.

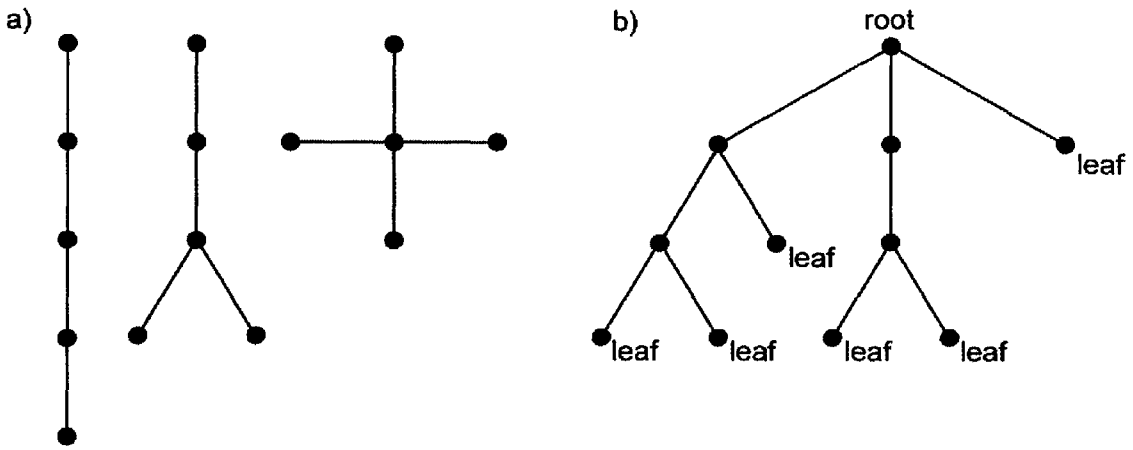

Figure 3.3 Trees: a) examples of non-isomorphic trees; b) a tree with a root and six leaves. (Worboys and Duckham, 2004)

Two graphs can be seemingly different – the distribution of vertices is different in the two graphs (geometry). However they can have the same connectivity relationships and such graphs are *isomorphic*. Sometimes it is difficult to detect that situation. For example, the graphs presented in Figure 3.4 are isomorphic – they have exactly the same number of vertices and edges, and the relationship between the vertices is the same.

The idea of duality is very important for this research. This can be explained using graph theory, but first planar graphs need to be explained. A *planar graph* can be drawn in a plane (on a piece of paper) with no crossing edges (edges intersect only at vertices): a graph (with crossing edges) is also planar if an isomorphic graph with no crossing edges exists. For example, graph G shown in Figure 3.5a is planar because it is possible to find an isomorphic graph G' that does not have crossing edges (see Figure 3.5b). However the order of edges, for

example, around vertex A is different: $AC \rightarrow AD \rightarrow AB$ in G, $AC \rightarrow AB \rightarrow AD$ in G' – the graphs are identically connected in a graph-theoretic sense, but not in the topological sense (Worboys and Duckham, 2004). An example of a non-planar graph is shown in Figure 3.5c.

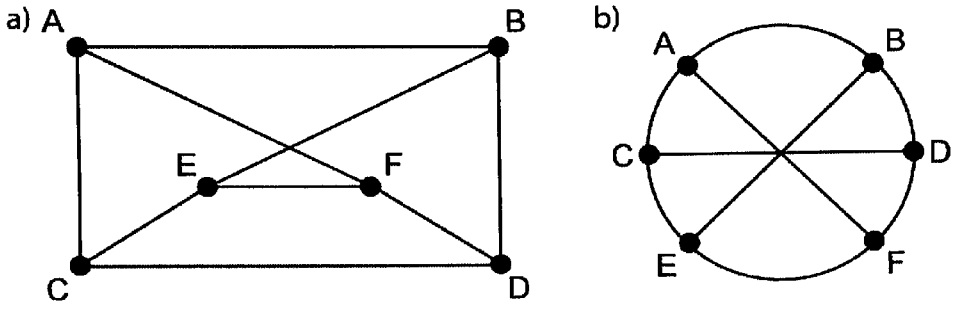

Figure 3.4 Isomorphic graphs: a) original graph; b) isomorphic graph – isomorphism can be difficult to detect. (Worboys and Duckham, 2004)

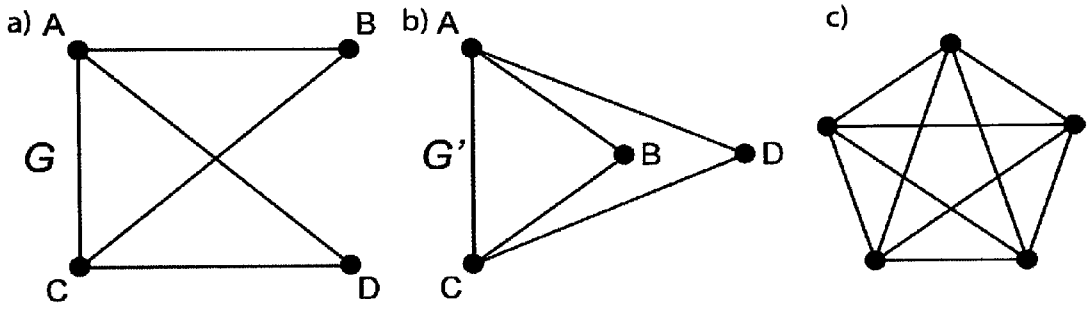

Figure 3.5 Planar and non-planar graphs: a) planar graph G with crossing edges; b) isomorphic planar graph G' with no crossing edges; c) non-planar graph.

If the planar graph is drawn without any edge intersections Euler's formula (Equation 3.1) is true:

$$v - e + f = 2, \tag{3.1}$$

where: v – the number of vertices, e – the number of edges, and f – the number of faces. A *face* is a region bounded by edges; the outer region with infinite surface area is also included. For example, there are four vertices, five edges, and three faces in the graph G' (Figure 3.5b) – Equation 3.1 is satisfied.

A *dual graph* G' of a planar graph G is obtained by assigning a vertex in G' to each face of G; two nodes in G' are connected with an edge only if the corresponding faces in G are adjacent (see Figure 3.6). In other words a dual vertex corresponds to an original face; a dual edge corresponds to an original edge. The original graph is called a *primal graph* (as opposed to a dual graph).

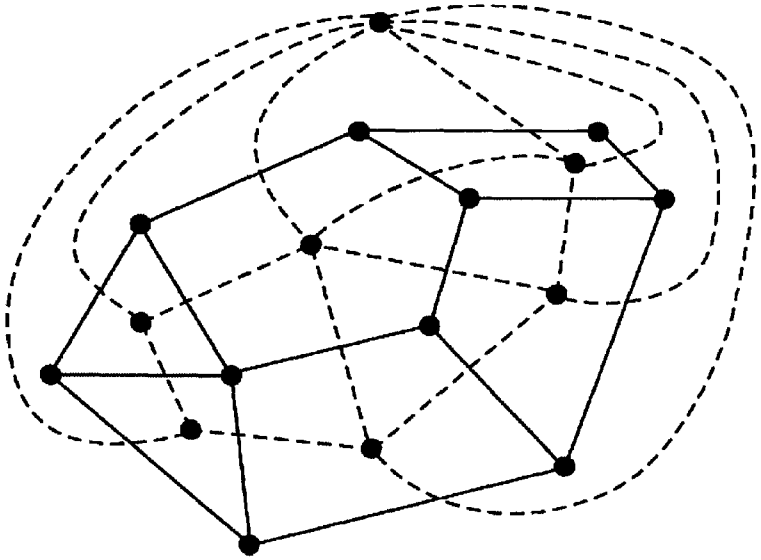

Figure 3.6 A dual graph G' (dashed lines) of a planar graph G (solid lines).

3.2 Duality

One of the most important structures, that are an example of duality, are the Delaunay triangulation (DT) and the Voronoi diagram (VD) (see Figure 3.7). The VD was formalized for n-dimensions by Voronoi (1908), but this is also known as: a Voronoi tessellation, a Voronoi decomposition, and a Dirichlet tessellation – Dirichlet used this structure (in two and three dimensions) in a study of quadratic forms (Dirichlet, 1850). The DT is a dual structure of the VD, and was formalized by Delaunay (1934) for n-dimensions. From the graph theoretical point of view the DT and the VD in 2D are planar graphs dual to each other.

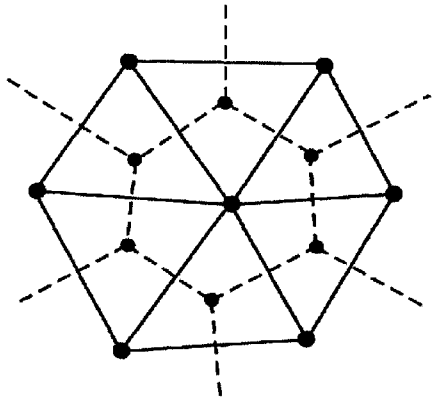

Figure 3.7 The Voronoi diagram (dashed lines) / the Delaunay triangulation (solid lines).

The VD (in 2D) is a decomposition of a plane into convex polygons (cells) such that each cell contains one generating point: every point inside the cell is closer to the generating point than any other. This structure is useful in applications that answer queries about the nearest

neighbours of a given point (i.e. the nearest hospital), in robots navigation to find clear routes, etc.

The DT (in 2D) is a triangulation (a decomposition of a plane into triangles) of a set of points in the plane such that there is no point inside the circumcircle of any triangle of the DT. Connecting all the centres of adjacent circumcircles using edges produces the VD. The DT is used for example in terrain modelling or to build unstructured meshes for the finite element method.

In three-dimensional space triangles are replaced with tetrahedra, and Voronoi polygons with polyhedra. A process of space decomposition into tetrahedra is called tetrahedralization. Duality rules conform to the 3D Poincaré duality rules (Munkres, 1984): for a space of dimension d and an element of dimension $k<=d$ a dual element exists of dimension d-k. Thus in 3D a vertex has a dual cell, a face has a dual penetrating edge, etc. (see Figure 3.8).

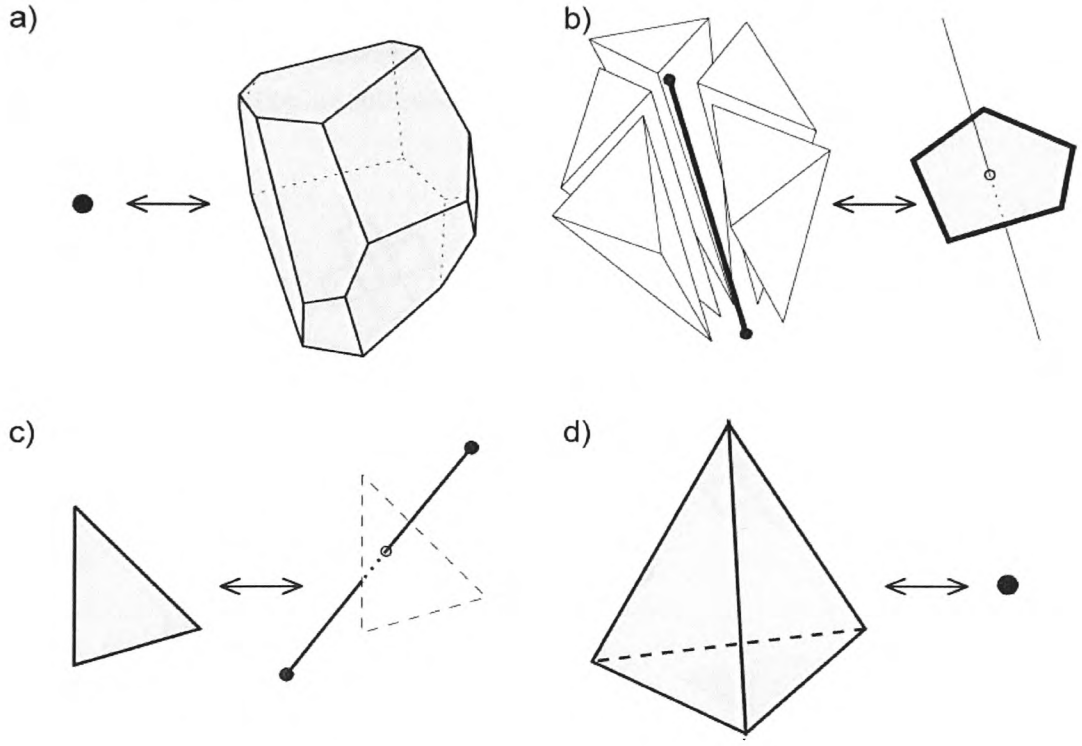

Figure 3.8 Poincaré duality. (Ledoux and Gold, 2007)

3.3 Data structures

In this section some examples of the data structures that are mathematically well described and can be used in various applications are presented. They are focused on a cell complex representation and allow for dual space maintenance (e.g. DT/VD dual representation). Other data structures typical for GIS and CAD are described in Sections 2.7 and 4.2 respectively.

The *Quad-edge (QE)* (Guibas and Stolfi, 1985) is a data structure developed to represent 2-manifolds – this simultaneously stores the primal and dual subdivisions. Each quad-edge consists of four quads and represents one geometrical edge and its dual edge. Each quad represents a directed edge that points to a vertex. It is important to remember that in 2D a dual vertex represents a primal face (and other way round – a primal vertex represents a dual face). There is no distinction between the primal and the dual space – they are symmetric (without an extra flag it is not possible to find out if one navigates in the primal or in the dual). Another advantage of the QE is that the structure is pointer based – all operations can have an algebraic representation. Each quad is represented by three pointers: *org* – the vertex associated with a directed edge; *next* – the next anticlockwise quad around the associated vertex; *rot* – the next anticlockwise quad in a quad-edge. These pointers connect all edges into a graph that permits navigation around shared vertices (a disc cycle) and faces (a face loop cycle): the second end of an edge is defined as *e.sym=e.rot.rot* (see Figure 3.9a). A construction process is also simple – there are only two operators: *MakeEdge* to create a new edge (quad-edge), and *Splice* to connect or disconnect edges (Splice is a self-reversible operator) (see Figure 3.9b).

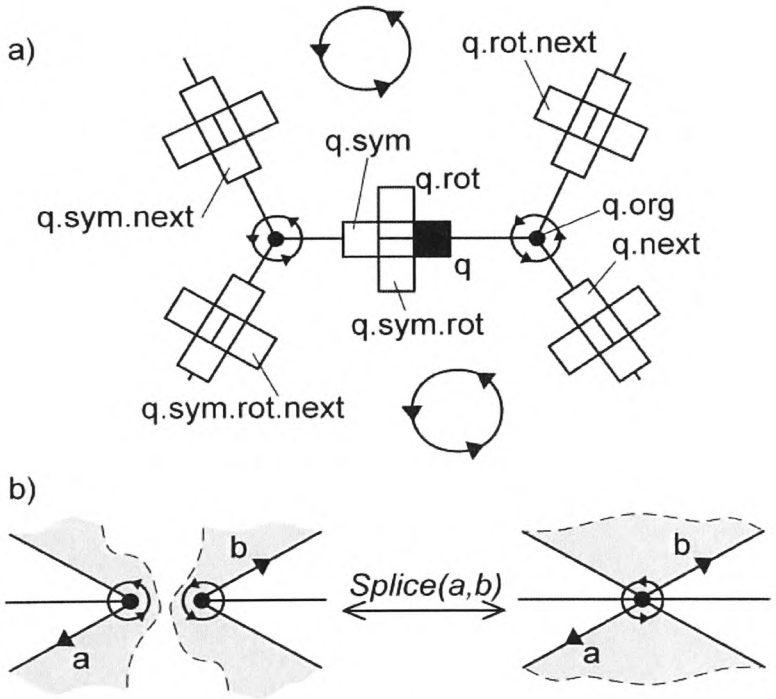

Figure 3.9 Quad-edge data structure: a) quads are connected by pointers; *q* (black square) is an original quad; b) the Splice operator to connect/disconnect edges.

The *Facet-edge* (Dobkin and Laszlo, 1987) is an extension of the quad-edge idea to the next dimension. To the best of the author's knowledge, this is one of the few data structures permitting the simultaneous representation of both the primal and dual subdivisions of a cell complex. It is based on face-edge pairs: part of an edge together with part of its associated face.

For every face-edge pair there are four facet edges, corresponding to the four ways of describing 'clockwise' directions within the face and around the edge. The facet-edge structure comes with a set of operations to modify cells and to navigate within both subdivisions. Its generality makes manipulation of a single cell too complex, and hence the authors suggest storing extra information for each edge and using the quad-edge operators. The facet-edge structure was defined to construct 3D subdivisions of space, thus all the cells of a complex are glued together by a face – there are no gaps or empty space between cells. The authors do not consider other adjacency relationships like two cells joined by an edge or a vertex. Also, to the best of the author's knowledge, it has never been fully implemented.

The *Generalized Map (G-Map)* structure was defined by Lienhardt (1991) for cellular models – which are generalizations of the B-Rep since each k-cell (i.e. 3-cell is a solid, 2-cell is a face, 1-cell is an edge, 0-cell is a vertex) is recursively decomposed into cells of lower dimensionality, and the topological relationships between adjacent k-cells are kept. For example, in two dimensions an atomic cell tuple might be a vertex "seen from" an edge "seen from" a face. All combinations of tuples are stored, which results in a space-consuming data structure. The G-map is very similar to the idea of the *cell tuple structure* (Brisson, 1993) that was developed independently (Mallet, 2002). The cell tuple structure is based on the quad-edge and the faced-edge data structures and allows for the simultaneous storage of the primal and the dual complexes (Brisson, 1993).

4 State of the art – CAD

Computer-aided design (CAD) is the technology for design using computer systems. The fundamental function of CAD is to define the detailed shape of the designer idea (Lee, 1999) – the geometry of the designed object. There are two main groups of CAD software: two-dimensional (2D) and three-dimensional (3D). The first one is a drafting system that uses lines/curves and figures to manipulate a shape of the object in two-dimensional space. The second one is a system for 3D geometric modelling – the shape of the object can be manipulated in three-dimensions using curves, surfaces and solids. Created models are mathematically described and can be used for further analysis. CAD is a general method where *solid modelling* plays a major role. Another crucial part is a graphic user-interface used to 'communicate' with the designer. Graphical feedback is generated immediately after the modification of a model. 3D interfaces should allow for model manipulation in the most intuitive way, which is not easy with a 2D display and input equipment. This chapter is focused on 3D and solid modelling representations.

The terms *2-manifold* and *non-manifold* are used quite often in this thesis. They should be explained in more detail. *2-manifold* is a surface (or exterior boundary) for which the neighbourhood of each point is topologically equivalent to a disc. In other words, each point of the surface divides space into two regions: inside and outside of the object; if any point does not divide space into two regions, then the object is *non-manifold.* The surface of a sphere (Figure 4.1a), a cube (Figure 4.1b), and a torus (Figure 4.1c) are examples of 2-manifolds: two polyhedra joined by a vertex (Figure 4.1d), edge (Figure 4.1e), or face (Figure 4.1f) are examples of non-manifolds. Every time the term manifold is used in this thesis it means the 2-manifold.

Another term that needs explanation is *homeomorphism* (or *topological transformation*). Disregarding mathematical definitions, it is easier to imagine homeomorphism as a transformation of an object as if it was made of rubber. It is possible to stretch or contract it but not to tear or fold it (Worboys and Duckham, 2004): topological relations like neighbourhood are preserved. For example, a sphere (Figure 4.1a) is homeomorphic to a box (Figure 4.1b) but not to a torus (Figure 4.1c).

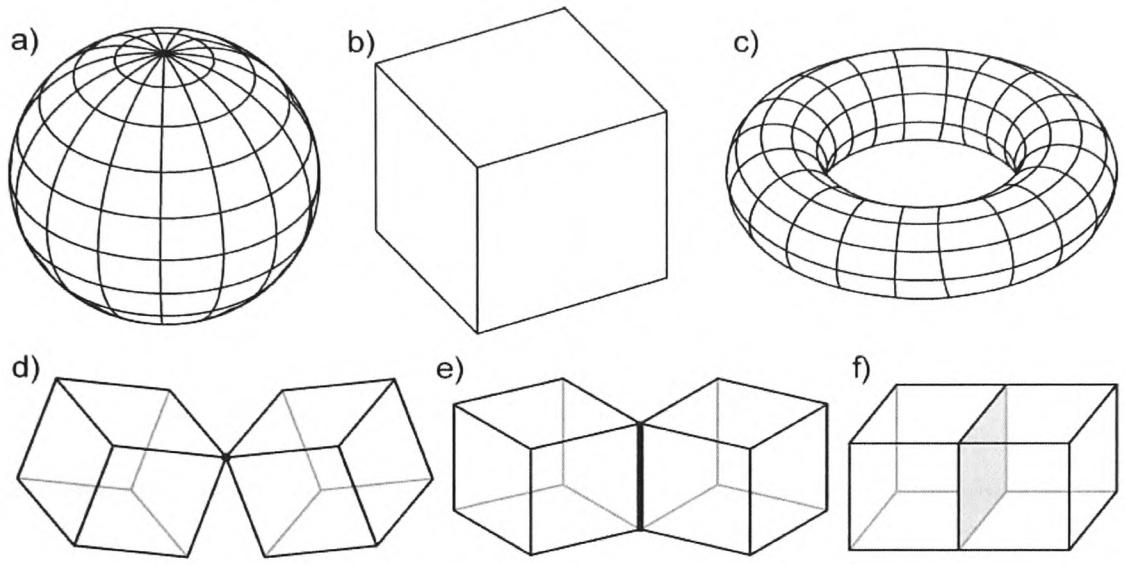

Figure 4.1 2-manifolds and non-manifolds: the surface of a) a sphere; b) a cube; c) a torus is a 2-manifold; two cubes joined by d) a vertex; e) an edge; f) a face form a non-manifold object.

4.1 Types

There are four classes of geometric modelling systems that are essential in 3D CAD: wireframe, surface, solid, and non-manifold modelling systems (Lee, 1999).

Wireframe modelling systems can be considered as a 3D extension of 2D drafting/graphical systems where only curves with their end points are used – called edges. A curve is described with equations and in a simple case can be a line segment. A model description contains a list of curve equations, coordinates of end points, and information about connectivity. The connectivity includes information about adjacency between edges and also associates curves with the points that are their ends. There is no information about the inside and outside boundary surfaces of the object. Thus it is not possible to unambiguously derive the faces of the object from its edges or calculate properties like mass or volume necessary for further analysis. Figure 4.2 illustrates an ambiguity in wireframe models – a set of edges is not sufficient to guess the polyhedra faces: is there a hole in the model or perhaps a complex of five, six or seven cells? The surface and solid models do not have these drawbacks and can replace

wireframe models. However a wireframe representation is one of the possible model visualization methods in 3D systems.

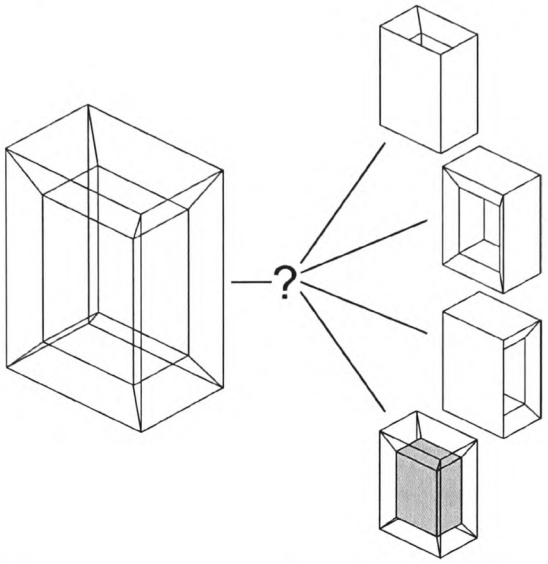

Figure 4.2. An ambiguous wireframe model. (Mäntylä, 1988)

Surface modelling systems are an extended version of the wireframe system. Information about surfaces and their shapes is also included in a model to make it geometrically complete. This allows for more advanced output (e.g. hidden line removal) or object collision detection. However free-form surfaces precisely describe the designed shape but the detecting and computing of intersections with other objects involves complex techniques and algorithms which make this method slow. Therefore planar faces are often used to create a tessellation of curved geometries that approximate the original shape. Usually the simplest polygon is used – a triangle: there are many simple and efficient algorithms for triangular mesh processing.

Connections between surfaces can be also added – otherwise surface boundaries have to be derived later by an application program.

Solid modelling systems are used to design shapes that have a closed volume (solids). Objects are defined by a boundary that is created by 'gluing' points, curves and surfaces together. The boundary of a solid is a 2-manifold. The principles of solid modelling are described by Mäntylä (1988). The main difference to the wireframe and surface modelling system is that non-closed volumes are prohibited (see Figure 4.3). A mathematical description of a model determines if any specified location is inside, outside, or on the boundary of a solid. The volume of a solid is a new entity that may be used in applications – any semantic information can be assigned not only to a surface but also to a volume. However construction of a solid in one step is too complex and not intuitive, thus the process is usually incremental: intermediate steps should be allowed – this is available with non-manifold models.

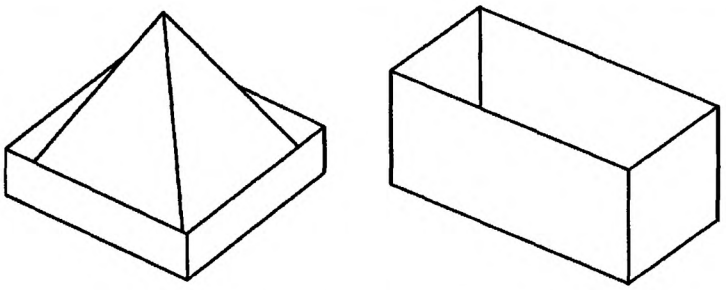

Figure 4.3 Invalid solids. (Mäntylä, 1988)

Non-manifold modelling systems allow for creating not only closed volumes but also open volumes (Figure 4.4a), and mixture of solids, surfaces, separate edges, and vertices (Figure 4.4b). Typical non-manifold examples are: two shells joined by a shared vertex (Figure 4.1d), an edge (Figure 4.1e), or a face (Figure 4.1f). In 2-manifold models an edge is coincident with exactly two faces, but in non-manifold models more than two faces can be joined to an edge. For example, in (Figure 4.1d) there is one common vertex for two shells; edges sharing this vertex create two separate cycles (one for each cell) which is not allowed for 2-manifold objects. The *cycle* is ordering information for a set of related entities. There are three kind of cycles in non-manifold modelling that have to be defined (Lee, 1999). The *loop cycle* is formed by edges around a face: this also corresponds to manifold models, and the cycle defined in graph theory (see Section 3.1). The *radial cycle* is formed by faces around a shared edge – in non-manifold models more than two faces meeting at an edge are allowed: in manifold models one edge is shared by only two faces. The *disc cycle* is formed by edges around a shared vertex. The difference between non-manifold and 2-manifold solid models is that a vertex may have more than one disc cycle in the non-manifold case, which is not allowed in solid modelling.

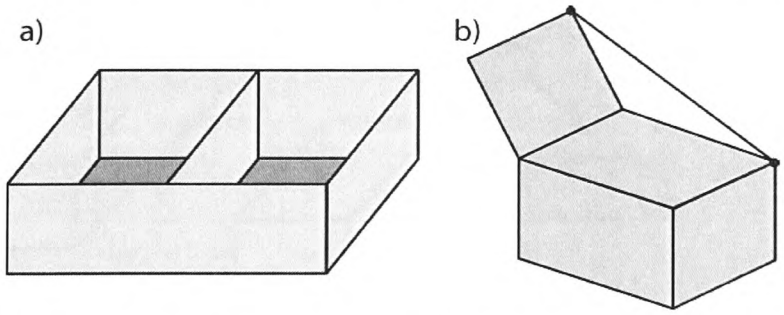

Figure 4.4 Non-manifold models: a) open volumes; b) a mixture of a solid, surfaces, edges and vertices. (Lee, 1999)

A non-manifold abstract model can be useful in the design process as an intermediate step. Dangling edges and laminar plates can then be extruded and thickness can be added to obtain closed volumes. Therefore the final version meets the solid modelling requirements.

4.2 Data structures

Regardless of the different modelling systems that can be used in CAD, the efficiency and ease of model modification are the most important issues. One of the problems is the mathematical representation of a solid in computer memory. Thus a data structure efficiently organizing the model in a computer is crucial. Different data structures are used in different applications depending on their purpose. However there are three main types of a data representation based on entities that need to be stored (Lee, 1999; Rossignac and Requicha, 1999): constructive solid geometry (CSG) representations, decomposition models, and boundary representations (B-Rep). In this section these data representations and some of the data structures common in CAD are presented.

4.2.1 Constructive solid geometry (CSG)

The constructive solid geometry representation uses primitives (simple shapes like cube, cylinder, etc.), specifies their size and position, and combines them together using the set of Boolean operators (union, intersection, and difference operators). A tree (CSG tree) is used as the main structure storing information about a solid. The tree is of a binary type, thus each node points at two lower level elements: another node or leaf. There are Boolean operators in the nodes of the tree, and the primitives used in the model are in the leaves. A simple example is presented in Figure 4.5: first the cylinder is subtracted from the big box, and then the small box is added. In other words the CSG tree is a history of applying the Boolean operators on the primitives.

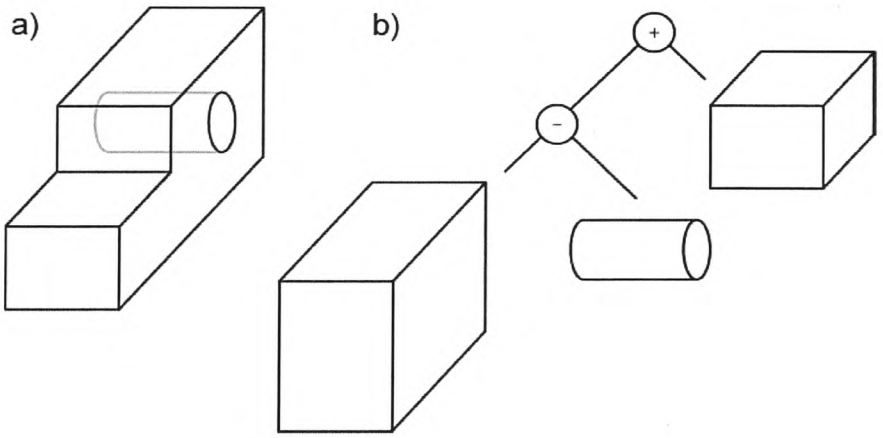

Figure 4.5 Example of the CSG representation: a) a complex object – a solid; b) the CSG tree with the Boolean operators in the nodes and the primitives in the leaves. (Lee, 1999)

The CSG representation is simple and data management is easy. However it does not explicitly carry any information on the connectivity or even the existence of the corresponding

solids (Rossignac and Requicha, 1999). A model is always valid because all the primitives are valid and the result of Boolean operators is valid. This representation can be converted to the B-Rep, but not vice versa – the inverse conversion is more difficult (Shapiro, 2002). There are also some disadvantages. Because a solid is represented as a history of the Boolean operators, local changes of shape can be very difficult or not possible to make. Computation of the solid boundary is complex and not efficient; the boundary surfaces, boundary edges, and connectivity between these elements are used for visualization of the solid. Because of these limitations, the B-Rep is sometimes added as a complement of the CSG representation. This blended solution is called the *hybrid representation*. However consistency between the two representations needs to be maintained which is not trivial. Usually a direct modification of a B-Rep model is not allowed – the conversion to the CSG representation is difficult and may cause a loss of consistency between models (Shapiro, 2002).

4.2.2 Decomposition model

The decomposition model represents an object as a set of non-overlapping connected cells that subdivide the space occupied by the object. There are three typical representations and data structures (Lee, 1999): the voxel representation, the octree representation, and the cell representation.

The *voxel representation* partitions space into a regular grid of cubes. First one large cube is created – this is the scope of operation. Then the cube is divided into small cubes (called voxels) by planes evenly distributed along the x, y, and z axis. This structure can easily be represented with a three-dimensional array. Each element of the array has a value: '1' if the cell is part of the modelled object, or '0' if the cell is outside of the object. Thus not only the object is included in the model, but also the external space. The voxel representation is accurate (as voxels can be divided several times to obtain the required accuracy): solids of complicated shape can be described precisely or only with a small approximation. This is a valuable property that allow for the building of models that is difficult using other methods; for example in medicine a human body and internal organs need to be modelled. The model accuracy can be increased by decreasing the size of a voxel. However together with the increase of the mesh density, the size of the required computer memory increases dramatically. Calculating mass or volume properties of the solid is easy; all the voxels are the same shape and size, thus it is not difficult to calculate the sum of voxels contained in the solid.

The *octree representation* is similar to the voxel representation. A cube is still the shape used in space partitioning, but the size of individual cells can be different. The original cube is incrementally divided along width, height, and depth planes into eight smaller cubes of the same size: in other words the original cube is divided into a 2×2×2 regular grid of cubes. Each of the

small cubes is called an octant because it is one-eighth of the original size. In the next iteration not every cube needs to be divided. Thus regions with the same properties (e.g. the interior of a solid or external space) are represented with bigger cubes, but regions that need to be modelled precisely (i.e. the boundary of a solid) are represented with smaller cubes that can be divided further until the required approximation is gained. Because the final grid is not regular (cubes can have different sizes) an array is not the optimal data structure. The octree structure is used instead – this is a tree where nodes represent the original cubes before the split, and each node has eight branches representing octants. The procedure for this method can be described in the following way: first the big cube enclosing the solid is created and is called the root octant; then this is divided into eight octants: if an octant is completely outside the solid it is marked as 'white', if it is completely inside it is marked as 'black', if it is partially inside and outside (the boundary of the solid) it is marked as 'grey'. White and black octants are final and are not divided further; in the next step only grey octants are divided until the required precision or size of octants is gained.

The *cell representation* aggregates simple cells to represent a solid as in the two previous methods, but in this case cells can have any shape. However in more restrictive models they have to be 2-manifolds (Shapiro, 2002) – topologically homeomorphic to a sphere (i.e. no holes). Manifold models are easier to represent, but on the other hand it is necessary to take into account that a union of two cells touching at a vertex or edge is non-manifold. Cells should be 'glued' together into one *cell complex*. Two cells can be connected along a face, edge, or vertex; they cannot overlap, but can be completely disjoint (Mäntylä, 1988). The incidence between cells can be calculated on demand from the geometry that has to be defined for every cell in the complex. But in practice this is not efficient (e.g. when the complex is traversed several times during analysis). Topological information about the connections between adjacent cells included in a model improves the efficiency significantly: no additional computation is required to answer topological queries. However including all incidence relationships increases the size of the data structure – which is impractical (Shapiro, 2002). Because the decomposition is not regular – does not form a regular grid, and there are many ways of representing connectivity between cells, there is no one optimal or universal data structure that could be used for the cell representation. This method is often used in a finite-element analysis.

4.2.3 Boundary representation (B-Rep)

The *boundary representation* stores information about the boundary of the modelled object. The basic entities necessary to build a model are: vertices, edges, faces, and solids – a solid is a volumetric element (polyhedron) bounded by a set of faces; a face is bounded by a set of edges; an edge is bounded by two vertices; a vertex represents a point in space. There is also

information about connections between the entities included in a data structure, which improves the efficiency of traversal operations and boundary processing. From a mathematical point of view the B-Rep data structure is a graph (Stroud, 2006); graph theory can be applied in many situations.

Relations between entities define the topology of a solid. The simplest representation is a set of three tables: vertices with their coordinates, edges with their end points/vertices, and faces with their bounding edges. This is very simple and compact but it is inefficient to obtain connectivity information or modify the model, when tables contain a large number of elements. A better idea is to use a different data representation. There are two typical data structures used in B-Rep for 2-manifolds: winged-edge and half-edge. Their biggest advantage is that connectivity between vertices, edges and faces are stored in the data structure. More complex solutions: radial-edge, partial-entity, coupling-entity, facet-edge, Generalized-Map, and Selective Geometric Complexes, allow for non-manifold modelling or cell complex construction.

The *winged-edge data structure* was described by Baumgart (1975) and this is probably the oldest data structure for B-Rrep. The main entity is an edge that stores topological information: two links to both vertices/ends of the edge (*PVT* and *NVT*), two links to both faces that share the edge (*PFACE* and *NFACE*), and four links to the next and previous edges around shared faces (*NCW*, *NCCW*, *PCW*, and *PCCW*) (see Figure 4.6). Because edges store information about both ends of the edge, and about the two faces connected to the edge, an edge can be considered as undirected – the direction of the edge is determined only by the order of the vertices. Thus it is necessary to check the orientation of the edge in the navigation process each time one navigates from one edge to another.

The *half-edge data structure* is based on the winged-edge and was used in the solid modelling system presented by Mäntylä (1988). The main difference is the splitting of an edge into two halves with opposite directions – he_1 associated with the V_1 vertex and he_2 associated with the V_2 vertex (see Figure 4.7a). Thus each half is directed and is a part of one face loop (an edge was shared between two faces in the winged-edge representation). The orientation of the loop is easy to determine without extra tests: the next pointer of the half-edge points at the next half-edge around the face in counter-clockwise direction looking from outside of the solid (*next_he*$_1$ and *next_he*$_2$ in Figure 4.7a) and the previous pointer points at the next half-edge in clockwise direction (*prev_he*$_1$ and *prev_he*$_2$ in Figure 4.7a). A face is defined as a loop of connected half-edges. However it is enough to associate a face with only one half-edge from the loop and the rest of the bounding half-edges can be derived when necessary. Navigation around a shared vertex is possible and is defined with the face loop connections (see Figure 4.7b): edges he_1, he_2 and he_3 share one vertex V_1; the first move to navigate from he_1 to he_2 is from he_1

to the previous half-edge (*prev_he*$_1$), then to the associated end of the edge (*he*$_2$); then the sequence is repeated but with *he*$_2$ as an origin half-edge; and so on.

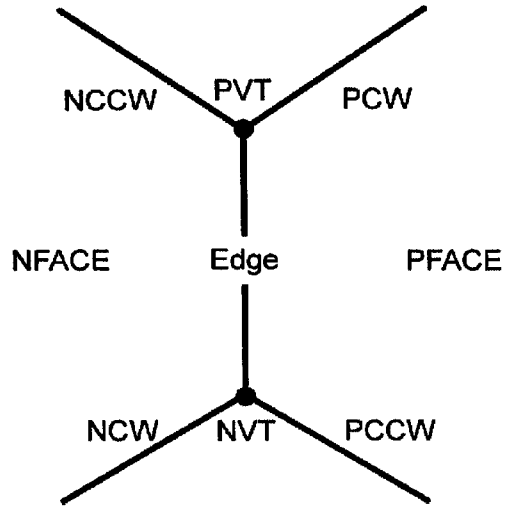

Figure 4.6 Winged-edge data structure; original link names are used (Baumgart, 1975).

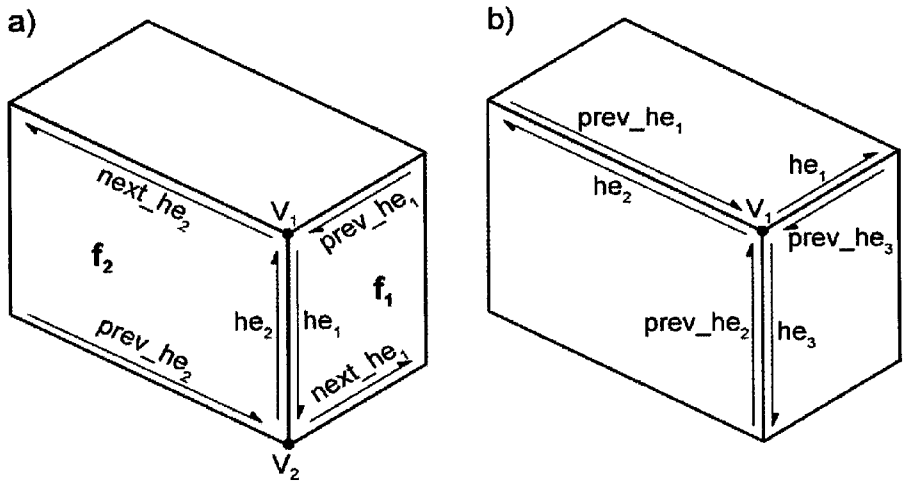

Figure 4.7 Half-edge data structure; a) a shared edge between faces f_1 and f_2 is split into two half-edges h_1 and h_2; the navigation between the two halves and around a face in both directions is possible; b) all edges of a solid are connected and navigation around a shared vertex is possible.

Holes in the faces of the solid are also available: each face can handle a list of pointers to one peripheral loop (the external boundary) and several hole loops (internal boundaries) (see Figure 4.8a); hole loops have the opposite direction to the peripheral loop. However another method can be used: instead of keeping several loops describing one face, loops can be connected into one external loop with *bridge edges* (see Figure 4.8b).

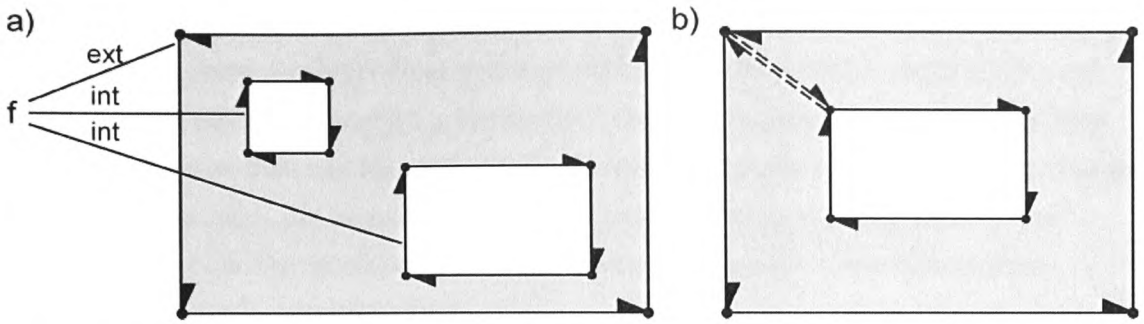

Figure 4.8 Face *f* with holes is represented as: a) one external (*ext*) and several internal loops (*int*); internal loops have the opposite direction to the external loop; b) one external loop with a hole are joined by the bridge edge (dashed lines).

The *radial-edge data structure* proposed by Weiler (1988) is the first complete representation of non-manifold B-Rep models (Lee, 1999). This is complex but is the most popular and efficient structure in many non-manifold systems (Lee and Lee, 2001). Two objects can be joined by a face, edge, or vertex. To represent these relationships new topological entities were introduced: face-use, loop-use, edge-use, and vertex-use. They are associated with the face, loop, edge, and vertex entities respectively. These 'uses' allow for representing objects as solids, as in solid modelling systems described in the previous section. Figure 4.9 shows the relations between edge-uses; two faces (f_1 and f_2) are joined and share an edge (see Figure 4.9a); this edge is represented by four edge-uses (eu_1, eu_2, eu_3, and eu_4) – one for each side of a two-sided face (see Figure 4.9b). Mate pointers connect two edge-uses located on the opposite sides of a face. Radial pointers connect two edge-uses located on radially adjacent sides of two faces. In this way, faces are ordered around a shared edge. There is also information about cycles stored in the data structure: a list of edge-uses in a loop (loop cycles), and a list of face-uses adjacent to an edge (radial cycles). Disc cycles (edges or faces around a shared vertex) are not stored explicitly – they are extracted from the geometric and topological data when required. Lee and Lee (2001) proposed the *partial-entity* structure that is a modification of the radial-edge. This compact representation reduces data storage by about a half without losing the time efficiency of the original structure.

The *coupling-entity data structure* was introduced by Yamaguchi et al. (1991) and Yamaguchi and Kimura (1995) for representing the non-manifold topology of three-dimensional B-Rep models. They call this new coupling-entity the *feather*. There are two groups of pointers in the structure: *mate pointers* and *cyclic pointers*. They describe relations between neighbouring entities. Three types of mate pointers point at other feathers in the model (see Figure 4.10a): fan mate (FM), blade mate (BM) and wedge mate (WM). The three cycles of feathers are defined: a disc – the next feather around a shared vertex, a loop – the next feather around a face, and a radial – the next feather around a shared edge. There is also a cycle

orientation taken into account: clockwise (C) and counter-clockwise (CC). Thus six cyclic pointers are necessary – counter-clockwise ones: disc (CCD), loop (CCL), radial (CCR), and clockwise ones: disc (CD), loop (CL), radial (CR). These cycle pointers can be deduced from the mate pointers as shown in Figure 4.10b (only counter-clockwise cycles are shown). Thus the full set of nine pointers can be reduced to three – mate pointers are used exclusively. The coupling-entity data structure is described in detail and compared to a new data structure developed during this research in Section 5.9.

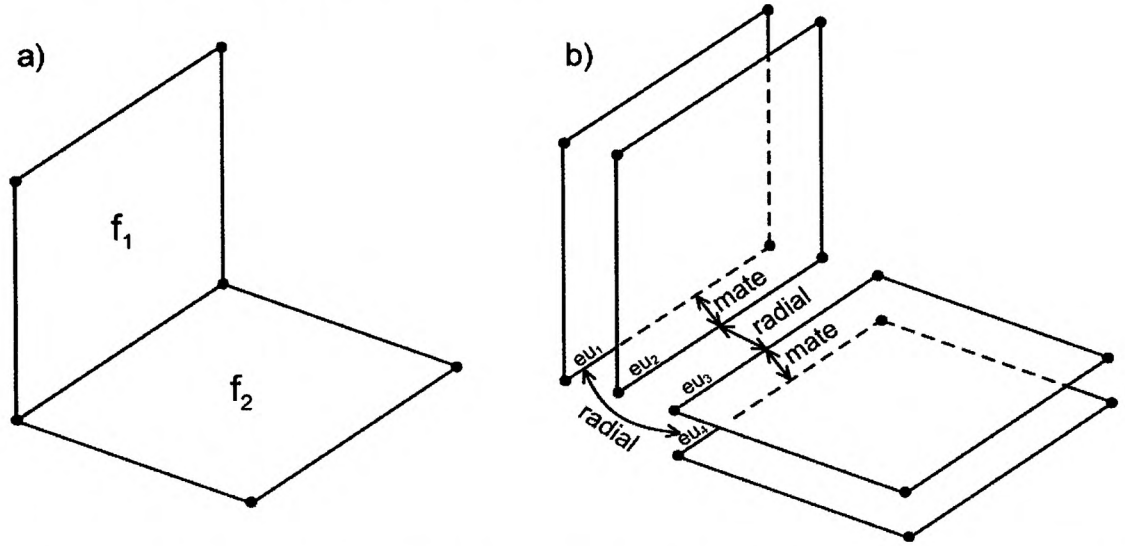

Figure 4.9 The radial-edge representation: a) two faces (f_1 and f_2) joined by a shared edge; b) mate and radial connections between four edge-uses (eu_1, eu_2, eu_3, and eu_4) representing the shared edge.

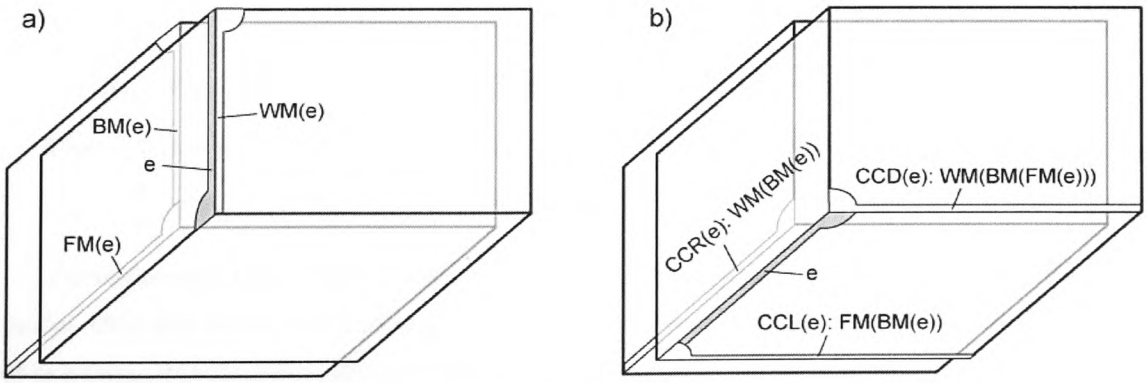

Figure 4.10 The feather entity: a) mates; b) cycles.

The *selective Geometric Complexes (SGC)* of Rossignac and O'Conner (1990) consist of a set of connected manifold cells (shells). The structure is based on incidence graphs (each vertex and edge of a B-Rep object is represented as a node in an *incidence graph*; these nodes are connected for every incidence between vertices and edges of the object); therefore there is no

ordering information as in the structures described in this section. SGC is not limited to 3D – a modelled object can have more than three dimensions; objects can also have an incomplete boundary.

4.3 Special representations

Originally B-Rep models are 2-manifold (Stroud, 2006) and are represented with data structures like the half-edge or winged-edge (described in the previous section). But sometimes models do not conform to the manifold condition: for example sometimes non-manifolds are produced in a modelling process. Therefore special cases must be taken into consideration and handled by a data structure. Stroud (2006) show three common types of special representations in B-Rep solid modelling: partial models, degenerate models, and non-manifold models (see Figure 4.11).

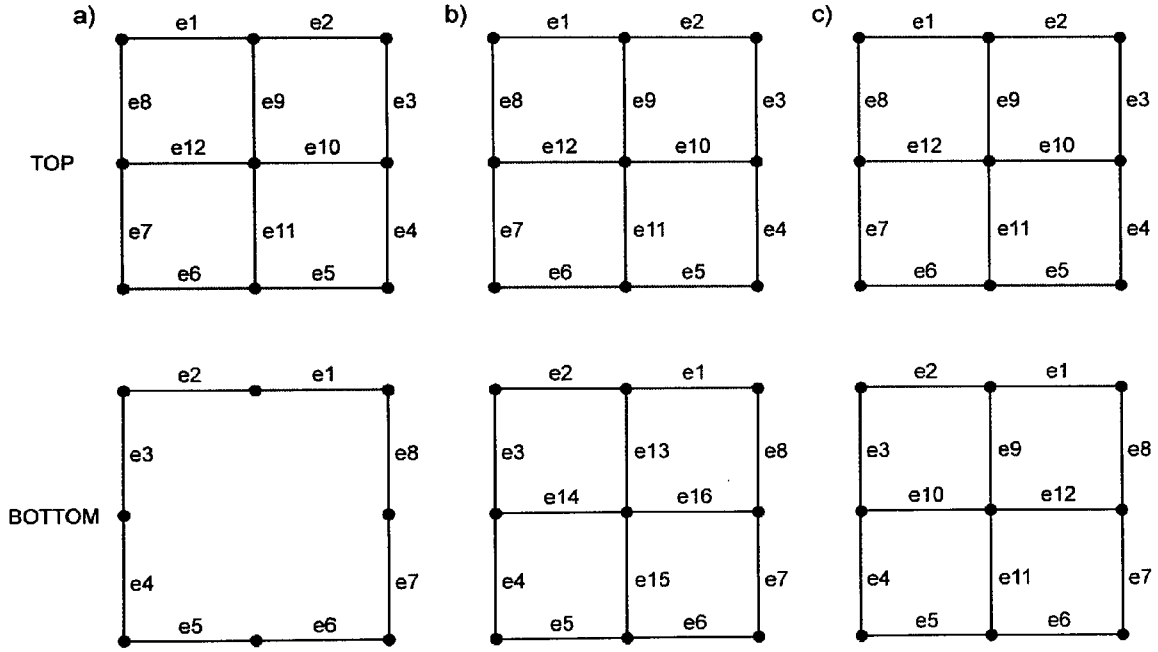

Figure 4.11 Special representations in B-Rep modelling: a) partial models; b) degenerate models; c) non-manifold models. (Stroud, 2006)

Partial models (see Figure 4.11a): The name comes from the partial completeness of a model. One side is fully defined and can be treated as locally manifold; the second one consists of one face, or is geometrically meaningless, or topologically undefined. Because only one side is defined these models are less computer memory consuming. They can be used to represent surfaces, or object parts, but their use is limited.

Degenerate models (see Figure 4.11b): A model is completely defined on both sides, locally manifold, and unambiguous, but larger than the other two types. Idealisations of thin plate models and temporary design stage representations are possible applications.

Non-manifold models (see Figure 4.11c): They are completely defined, as the degenerate models, but all adjacent edges and vertices are merged. Therefore, an edge can be shared by more than two faces (which is not allowed in manifold models). Compound objects for process planning or FEM models can be represented with this kind of model.

Partial and degenerate models can use standard B-Rep data structures (e.g. the half-edge, the winged-edge), while non-manifold models needs more complex data structures (e.g. the radial-edge). To avoid using complex representations a model can be converted. One method is to approximate a non-manifold model with a manifold model (Rossignac and Cardoze, 1999) – models are infinitely close one to another in the geometric sense. For example, two boxes sharing an edge (shown in Figure 4.12a) is a non-manifold case that after some conversions can be modelled as 2-manifold shell(s): the shared edge is duplicated – the boxes are represented as two shells (see Figure 4.12b); the second method duplicates the shared edge too, but boxes are 'merged' to form one solid (see Figure 4.12c) – the final result is an example of the degenerate model. This conversion permits one to use a manifold data structure to represent non-manifolds: that model is sometimes called a *pseudo-manifold representation.*

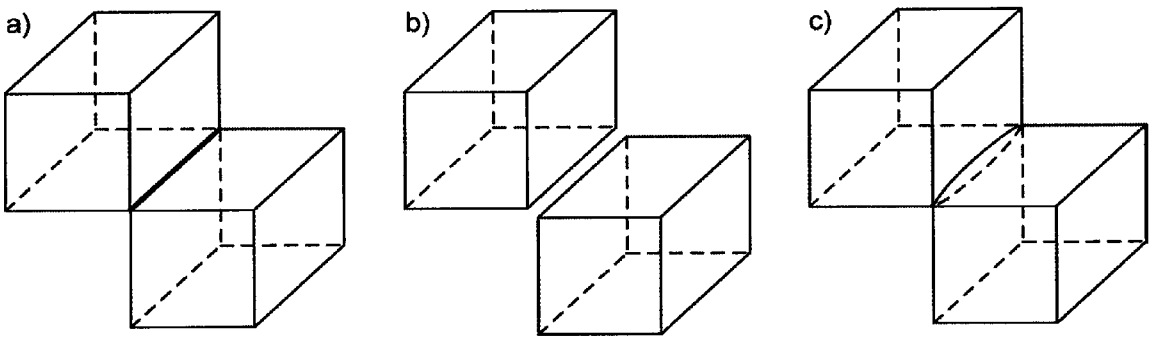

Figure 4.12 Non-manifold to 2-manifold conversion: a) non-manifold model – two boxes sharing an edge; b) boxes can be represented as two shells; c) boxes can be represented as one solid.

4.4 Euler operators

During the model building process a bulk model is often built automatically, and then modified or corrected later. Alternatively, starting with an initial shell creation, the model is built by incremental addition of individual segments. Thus the data stored on the computer needs to be modified. At the lowest level there are operations directly changing the data structure: their implementation depends on the data structure properties (fields, pointers, etc). However it is possible to create a set of standard operators (that uses lower levels or changes the data directly) that could be used in applications without a knowledge of the data structure details. In CAD, where B-Rep solid modelling is widely used, such standardized operators are called *Euler operators*. They change a model at the level of B-Rep entities: shell, face, loop, edge, and

vertex. For example there are operators: to make an edge, to split a loop, delete a vertex, and many more. They have to satisfy a series of rules (Braid et al., 1980) (Equations 4.1 – 4.4): Equation 4.1 means that a negative number of entities is prohibited; Equation 4.2 means that solids and holes through solids are prohibited if no other entities are present in a model; Equation 4.3 means that a valid object consists of at least one vertex, and at least one face; the B-Rep model (which is a collection of faces, edges, and vertices) is valid if the numbers of entities satisfy the *Euler-Poincaré formula* (Equation 4.4).

4.1 $$v, e, f, h, g, s \geq 0$$

4.2 $$if\ v = e = f = h = 0, then\ g = s = 0$$

4.3 $$if\ s > 0\ then\ v \geq s\ and\ f \geq s$$

4.4 $$v - e + f - h = 2(s - g)$$

where, v is the number of vertices, e – the number of edges, f – the number of faces or peripheral loops, h – the number of hole loops, s – the number of shells, and g – genus (the number of handles – holes through a solid; for example a sphere has genus 0, a torus has genus 1). Because all these six entities cannot be independently manipulated (only five of them can be) it is easier to manipulate entities in small groups than separately; and the minimal number of necessary operators is five (Lee, 1999). A set of five Euler operators that can be used to create or modify the topology of a shell is called a *spanning set* (Braid et al., 1980). A set is spanning if each one of the six entities can be modified by at least one operator. This is also true for each of their inverses. One of the possible spanning sets (Stroud, 2006):

MEV – Make an Edge and a Vertex

MEF – Make an Edge and a Face

MBFV – Make a Body (new shell), Face and Vertex

MGB – Increase the Genus and Make a Body (shell)

MEKH – Make an Edge and Kill a Hole

Each of these operators changes the number of any entity by at most one. For example, MEV changes the number of vertices and edges by one, and does not change the number of faces, holes, shells or genus, which can be written as a vector (+1, +1, 0, 0, 0, 0). All operators of the spanning set can be written as a matrix M:

4.5 $$M = \begin{matrix} \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} v & e & f & h & s & g \\ \end{matrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 1 & -1 & 2 & -2 \end{bmatrix} \begin{matrix} \text{MEV} \\ \text{MEF} \\ \text{MBFV} \\ \text{MGB} \\ \text{MEKH} \\ \\ \end{matrix}$$

where columns represent in order: the number of vertices, edges, faces, holes, shells, genus to change. Rows correspond to operators from the spanning set, and the last row corresponds to the coefficients of the Euler-Poincaré formula (which acts as the normal vector of the formula).

Mäntylä (1988) states that the number of entities $\{v, e, f, h, s, g\}$ in each data structure will satisfy the Euler-Poincaré formula (see Equation 4.4). Therefore the addition of any entities by a single Euler Operator will preserve this relationship: examining the row representing each operator in matrix M (see Equation 4.5) will show that this is true for each operator: for example for MEV *1v-1e=0*. Thus for any model the number of Euler Operators from a spanning set that are required to build the model and preserve the Euler-Poincaré formula needs to be found.

This may be written as

4.6 $$q = pM$$

where q is a vector of the number of each of the six types of entities and p is a vector of the number of Euler Operators needed. If these six conditions can be satisfied (the five selected Euler Operators plus the Euler-Poincaré formula) then the structure can be constructed with the number of operators given by q. (The first five elements give the number of times each Operator is required. The sixth element, the number of times the Euler-Poincaré formula is applied, must remain zero as it is a basic condition for a valid structure.) In order to find p, the number of Euler Operators needed, the inverse of M is required:

4.7 $$M^{-1} = {}^{1}/_{12}\begin{bmatrix} 7 & -5 & 4 & -2 & -1 & 1 \\ 5 & 5 & -4 & 2 & 1 & -1 \\ -5 & 7 & 4 & -2 & -1 & 1 \\ 5 & 5 & -4 & 2 & -11 & -1 \\ 2 & 2 & -4 & 8 & -2 & 2 \\ -2 & -2 & 4 & 4 & 2 & -2 \end{bmatrix}$$

For the example of a cube $q=\{8, 12, 6, 0, 0, 1\}$ (8 vertices, 12 edges, 6 faces, 0 holes, genus 0, and 1 shell) p can be calculated from Equations 4.6 and 4.7: $p=\{7, 5, 1, 0, 0, 0\}$ – the cube can be constructed with seven MEV, five MFE, and one MBFV operators. The MBFV has to be used as the first, because this is the only operator that creates a shell (body): vertices, edges, and faces cannot exist without a shell. The sixth zero shows that the Euler-Poincaré formula is satisfied.

There are different possible strategies for cube construction – MEV and MFE operators can be arranged in various sequences. Figure 4.13 presents two scenarios, but there are more possible: Figure 4.13a shows a case where all seven MEV operators are used first before any MEF operator; Figure 4.13b shows a more 'natural' or intuitive method.

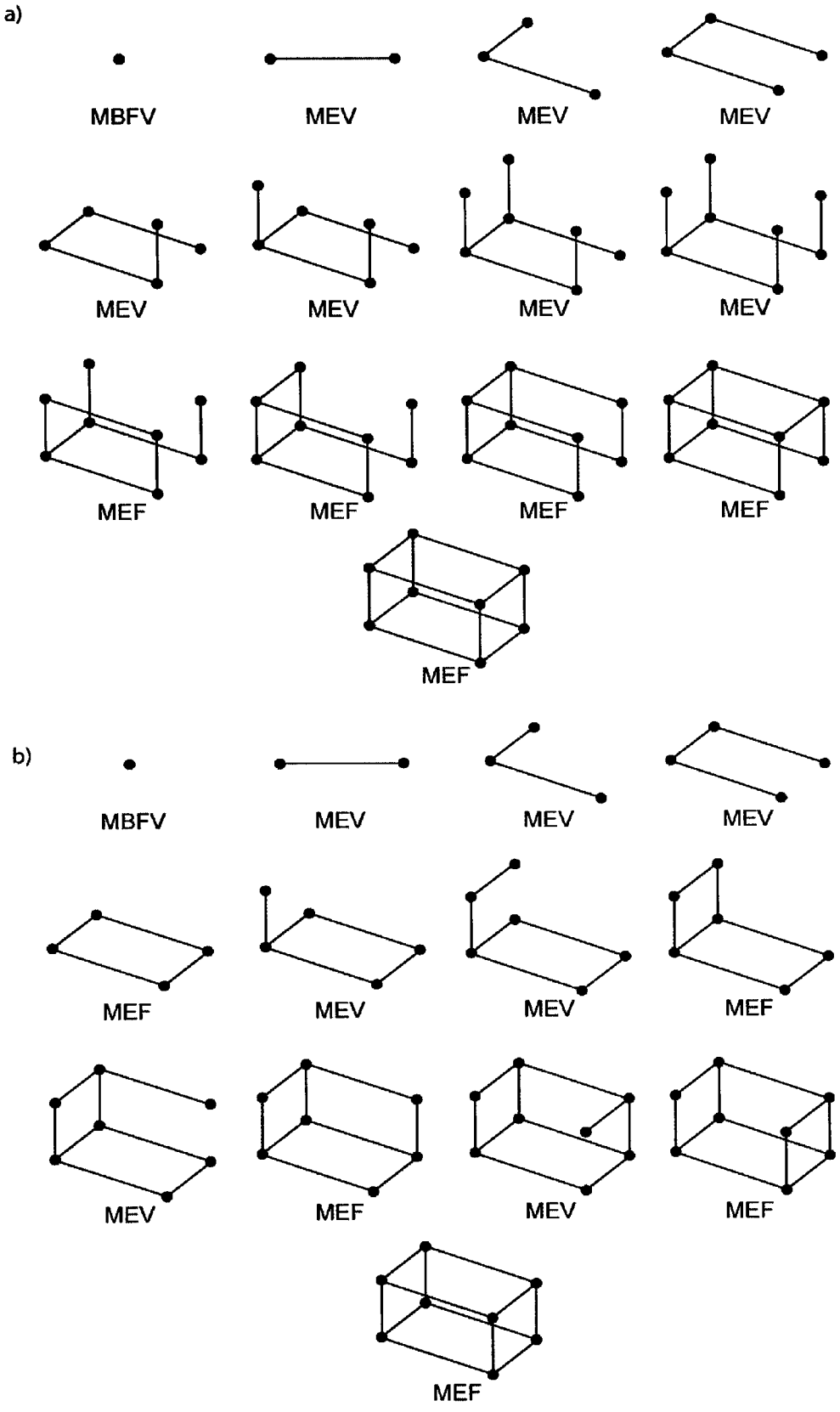

Figure 4.13 Cube construction scenarios using Euler operators (MBFV is always first): a) MEV operators are used before MEFs; b) a cube is constructed face-by-face.

Euler operators are accompanied by reverse operators. For example, MEV (Make an Edge and a Vertex) is paired with KEV (Kill an Edge and a Vertex); MEF (Make an Edge and a Face) – KEF (Kill an Edge and a Face). The reverse operator can undo what was made by the original operator. They can be used to cancel operations that were performed in the design process. However, to cancel those operations that have been done at the beginning, all the following operations have to be cancelled. This can be compared to the Undo-Redo operations in word processor applications.

For practical reasons there are usually extra Euler operators developed in a final implementation. They allow for simpler changes than just a spanning set. Stroud (2006) lists all 99 possible Euler operators (that change the number of entities by one), and chooses 39 as the most useful that can be used in an application programming interface or a design tool. However Euler operators are simple, and sometimes geometric tests are necessary to determine the orientation of entities before an Euler operator is performed. These tests are also important because Euler operators change the topology and they can produce geometrically incorrect objects.

It is also interesting to notice that some Euler operators can have different interpretations. For example MEV, which produces one vertex and one edge, can: 'split' a vertex and add an edge in between (Figure 4.14a); just add an edge with a vertex to an existing structure (Figure 4.14b); or 'split' and edge with a new vertex (Figure 4.14c).

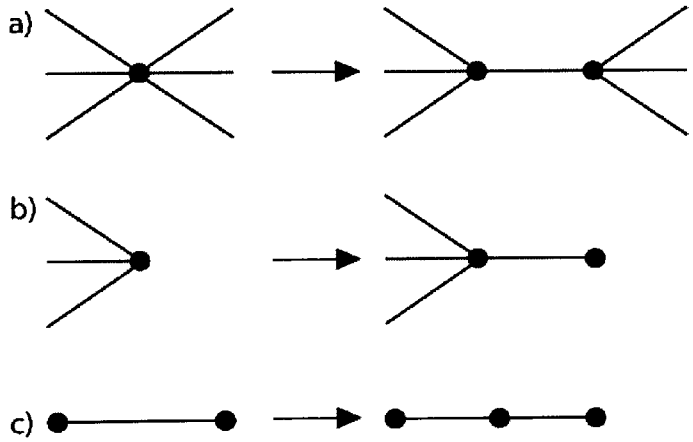

Figure 4.14 Possible interpretations of the MEV operator: a) 'split' a vertex and add an edge in between; b) add an edge to an existing structure; c) 'split' an edge with a new vertex.

There is an open question how to integrate the three main elements of B-Rep models: geometry, topology and information: Euler operators affect the topology only (Stroud, 2006). One of the solutions is to implement construction operations as layers with topological Euler operators at the lowest level: the geometry and information can be handled in the higher layers.

The 'standard' Euler operators presented above apply to single closed shells. They are not suitable for a cell complex or non-manifold construction. *Extended Euler operators* were proposed for non-manifold modelling including cell complexes (Masuda, 1993; Masuda et al., 1989). In this case a spanning set consists of nine operators. The Euler-Poincaré formula for non-manifold models is described by the Equation 4.8:

4.8 $$v - e + f - h - (V - V_h + V_c) = C - C_h + C_c,$$

where: v – the number of vertices; e – the number of edges, f – the number of faces, h – the number of holes in faces, V – the number of volumes (cells in a complex), V_h – the number of holes through volumes; V_c – the number of cavities in volumes; C – the number of complexes; C_h – the number of holes through complexes; C_c – the number of cavities in complexes. For example, the parameters for a figure shown in Figure 4.15 are as follow: v=16, e=24, f=12, h=2, V=2, V_h=1, V_c=0, C=1, C_h=0, C_c=0; the formula 4.8 is fulfilled. In this example the object is represented with one cell complex (C=1) consisted of two boxes (V=2); there is one big box with a hole through it (V_h=1) – V_1; the hole is filled with the second box – V_2.

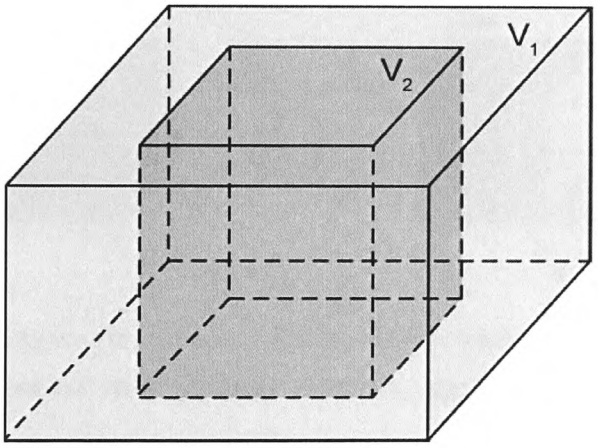

Figure 4.15 An object represented with a cell complex consisted of two boxes – the smaller box V_2 is put into a hole through the bigger box V_1.

The formula 4.8 is also valid for solid, surface, and wireframe models (Masuda et al., 1989). In solid modelling cell complexes are not permitted – an object is represented as one volume; therefore V=C, V_h=C_h, and V_c=C_c; and also s=C+C_c, and g=V_h. In this way Equation 4.4 is obtained. In surface modelling, there are no volumes, therefore V=V_h=V_c =0, thus the following formula (4.9) can be applied:

4.9 $$v - e + f - h = C - C_h + C_c$$

In wireframe modelling there are no volumes and no faces in objects, therefore f=h=V=V_h=V_c=C_c =0; thus Equation 4.10 is valid:

4.10 $$v - e = C - C_h$$

It should be noted that Equation 4.8 is data structure independent (Masuda et al., 1989) and can be interpreted in a different way. For example, a cube with a cavity can be one volume with empty space inside (the cavity) (V=1, V_c=1, C=1, C_c=1), or it can be a cell complex of two cubes – the smaller cube represents the cavity inside the bigger cube (V=2, V_c=1, C=1, C_c=0). In both cases the Euler-Poincaré formula is fulfilled. In the case that cavities are represented as volumes (not empty spaces) the number of operators in the spanning set is eight, because it is no longer necessary to explicitly represent cavity entities – they are represented as cells of a complex. This fact is important in this research, because cell complexes that decompose space are constructed, and each fragment of space is represented as a cell (there is no empty space in the models).

Some examples of the extended Euler operators used were presented by Masuda (1993). One of them is the *split_volume* operator (see Figure 4.16) which splits a volume into two by adding a face that 'cuts' the volume in two. This face is shared by the volumes, thus the resulting model is non-manifold. The reverse operator *merge_volume* removes the face to merge the two adjacent volumes.

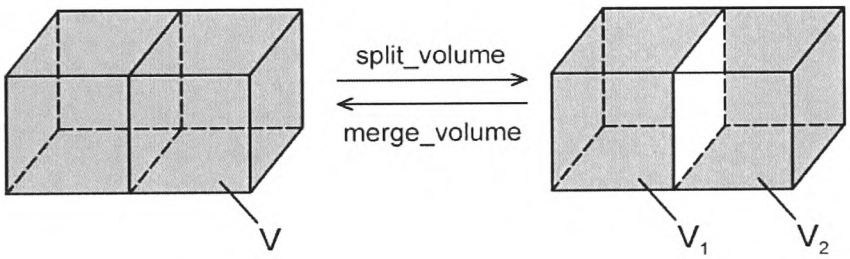

Figure 4.16 An example of the extended Euler operators introduced by Masuda (1993): *split_volume* splits volume V in two volumes V_1 and V_2; *merge_volume* is a reverse operator.

Masuda (1993) also presented an example of a lift operation based on a sequence of Euler operators. The lift operation is used to extrude 3D objects from their 2D footprints. The process is shown in Figure 4.17. A face and a wire edge are an input 2D model (see Figure 4.17a). Wire edges are generated from the existing vertices (see Figure 4.17b): other wire edges are generated between the newly created vertices (see Figure 4.17c). In the next steps five side faces are generated (see Figure 4.17d) and one closing face on a top of the empty box (see Figure 4.17e). In the last step a volume is defined (see Figure 4.17f). The result is a non-manifold model – a solid (a box) with a laminar face attached.

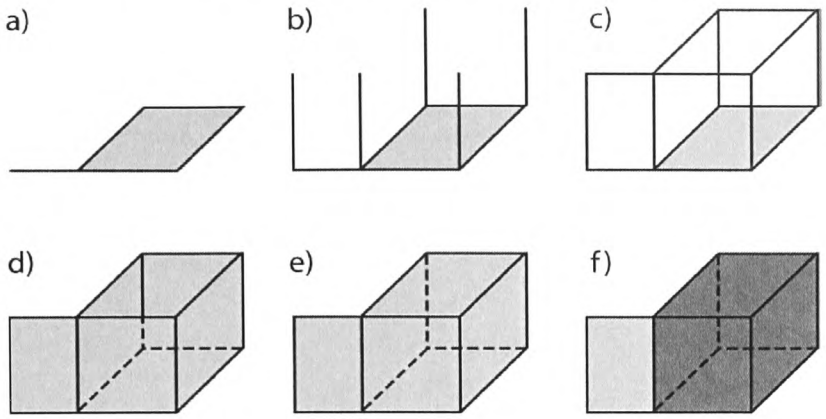

Figure 4.17 Lifting a face and a wire edge. (Masuda, 1993)

5 The dual-half-edge (DHE) data structure

This chapter presents the main research results. A new data structure and construction operators were developed during the project – a detailed description, modified versions and comparisons with different data structures are included.

5.1 Problem area

3D models are highly desired as they improve simulation efficiency (Kwan and Lee, 2005). They allow for a more accurate phenomenon or object representation. Modern simulation systems operate on bigger and more detailed models. They demand more efficient algorithms and structures for analysis. Fortunately technological developments provide faster computers that allow for complex simulations not possible a few years ago. Non-manifold B-Rep models (see Section 4) are currently very popular (Lee and Lee, 2001): they allow for more flexible modelling than the standard solid modelling.

However CAD structures do not fit the single-shell topological model assumed in GIS. Instead of the usual flat map concept the whole earth can be considered as a single polyhedron – the "Polyhedral Earth" (Tse and Gold, 2004). This could be added to a normal GIS as an extension of TIN modelling. Typically in GIS single-shell building models are simply positioned geometrically on the ground surface – this prohibits advanced analysis such as flow over the ground surface and around buildings and through tunnels.

For full 3D modelling with multiple volumes (e.g. building interiors, rooms) a different (new) data structure is needed. 3D cell complexes are effectively adopted as mathematical (topological) models for 3D non-manifold objects (Dobkin and Laszlo, 1987; Lee and Lee, 2001; Masuda, 1993).

5.2 Objective

The main objective in the project was to develop a new data structure that permits one to build full 3D object models, in particular models of building interiors that can be used in emergency management systems. There are two important things that should be possible to manage: geometry – to visualise a model, and topology – to implement some algorithms required in order to know connections between rooms in a building. It should be also possible to store them simultaneously in two graphs: primal and dual; the primal graph may be identified as the geometry of a model, while the dual – as the topology. Another important issue was to have an elegant, CAD-like way of object construction if possible. Assigning of attributes should be possible: attributes may be used to store the semantic information of a model. The author could not find a data structure with these properties.

5.3 Background

The DHE builds upon previous work of the group lead by Professor Gold on a structure called the augmented quad-edge (AQE) (Ledoux, 2006; Ledoux and Gold, 2007). The AQE uses the quad-edge (QE) data structure (Guibas and Stolfi, 1985) to individually represent each polyhedron. With this structure, it is possible to navigate within a single cell with the quad-edge operators, but it was initially impossible to navigate to adjacent cells. That problem was solved by using the dual to link pairs of adjacent polyhedra. Thus the ability to navigate the primal and dual graphs of a single edge using the quad-edge approach is preserved, and the 3D dual edge forms part of a complete dual cell complex with exactly the same structure. Ledoux and Gold (2007) showed that this structure provided navigation for 3D Voronoi/Delaunay structures, however the construction operators were complex and arbitrary 3D models and non-manifold cases were not supported.

The DHE data structure is a modification of the AQE (Boguslawski and Gold, 2008) and is related to the facet-edge (Dobkin and Laszlo, 1987), radial-edge (Weiler, 1988) and half-edge (Mäntylä, 1988) structures.

5.4 Properties/overview

5.4.1 The external cell

In the proposed representation a model is composed of a cell complex – it can be a building with rooms represented as cells, or a machine part decomposed into smaller cells that can be used in the finite element method. Each cell of the complex is a 2-manifold. There is always an external cell present in the model that encloses the rest of cells in the complex. While the internal cells represent modelled objects, the external cell can be imagined as 'the rest of the

world'. Therefore its volume is infinite. This external cell is required because all cells in the complex are connected by an adjacent (common) face; cells located at the boundary of the model would not have an adjacent cell to connect to, and thus the topological consistency would be invalid. By adding the external cell, the navigation in the complex (from cell to cell) can be performed without testing if a boundary of a model is approached (i.e. not to fall off the edge of the world).

5.4.2 Components (entities) of the model

The entities present in the model are: cells, faces, edges, and vertices. The *cell* is a 3D B-Rep shell with a zero or positive volume. Note that the volume can be zero if, for example, a cell consists of two identical faces joined together by the same set of edges. A cell is bounded by faces that form a closed shell. *Faces* are convex or concave polygons – there is no restriction to triangle faces.

It is assumed that faces are flat and their flatness is not tested during the construction process. However non-flat faces appear at intermediate steps of the process. In principle they are also allowed in the final model, but additional tests would be necessary, for example to check the adjacency of two non-flat faces before two cells are joined.

A face is bounded by edges connected into a loop (they form a loop cycle). Each *edge* is terminated with two vertices. It is assumed that an edge is a straight line segment. It is possible to define an edge as a curve, but this would need extra developments, as in the case of non-flat faces. An edge is represented by two connected DHEs (details are presented below).

A *vertex* represents a point in 3D space and does not store any topological information. Each unique vertex (with unique coordinates) is stored only once in a memory: a reference to the vertex is used where required. For example, several edges can share one vertex, but the vertex is not duplicated and assigned to each edge – only references are used to connect the vertex with edges.

5.4.3 Duality of the model

The presented representation has a dual nature, with complete symmetry between the two structures. There are two spaces (two structures): the primal and the dual, and they are represented as graphs. Each entity (volume, face, edge, and vertex) in one space is matched by another entity in the dual space. It conforms to the 3D Poincaré duality rules: for a space of dimension d and an element of dimension $k<=d$ a dual element exists of dimension $d-k$. Thus in 3D a primary vertex is equivalent to an enclosing dual cell, a primary face to a dual penetrating edge, etc. (see Figure 3.8) This reduces the number of entities in the representation to two: edges and vertices. It must be emphasized that this equivalence applies in both directions: a

primary cell is referenced by its dual vertex, a primary face by its dual edge – and dual faces and cells, if needed, are represented by their primal edges and vertices.

A cell can be represented as a single dual vertex – there is no need to create a special class of objects to represent a cell: properties of a cell (e.g. ID, volume) can be assigned to its dual vertex.

Adjacent cells of a complex are connected by a shared face, which is represented by a dual edge (see Figure 5.1a). This edge links two dual vertices representing the adjacent cells. Technically, each face is penetrated by a bundle of dual edges – the number of dual edges is the same as the number of edges forming a face. For example, each face of a single cube cell is represented as a loop of four half-edges; each half-edge has an associated half-edge in the dual; thus each square face is penetrated by a bundle of four half-edges in the dual (see Figure 5.1b). Each one of these dual half-edges belongs to the dual cell surrounding one of the four vertices of the face (see Figure 5.1c). Navigation around a bundle (radial cycle) is possible – the first step is to navigate to a dual space, go around a face loop, and go back to the original space.

A face in terms of the Poincaré duality is represented in the model as a bundle of edges that belongs to several cells sharing that bundle (see Figure 3.8b and Figure 5.1).

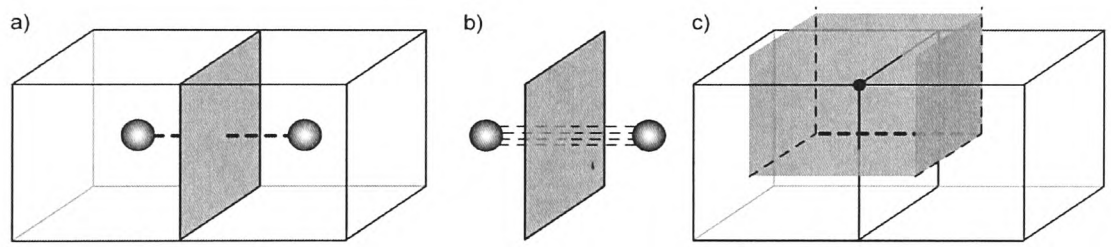

Figure 5.1 Connections between cells: a) adjacent cells, that share a face (grey), are connected by a dual edge (dashed line); b) a dual edge is represented in the model as a bundle of edges penetrating a face; c) each one of the edges from the bundle belongs to a cell surrounding one of the face vertices.

From a navigational point of view, primal and dual spaces are identical – without an extra flag it is not possible to tell which space is navigated. Navigation in these two structures, and thus traversing the primal and the dual graph, is the same. The primal structure is usually associated with the geometry of a model, while the dual structure represents the topology of a cell complex (the connections between cells).

5.4.4 Holes and cavities

Another important property of the models is that holes and cavities are allowed: a 'bridge edge' (e_b in Figure 5.2a) is used to connect internal rings (holes) to the outside ring (the face) (f_{INT} and f_{OUT} respectively in Figure 5.2a); and a 'bridge face' (f_b in Figure 5.2b) to connect an internal

cell (a cavity) with the outside cell (c_{INT} and c_{OUT} respectively in Figure 5.2b). Bridge edges and bridge faces are added to a model as other edges and faces using the same operators described later. The only difference is that a bridge edge has a special attribute that is taken into consideration by the navigation operators. This edge can be omitted during the navigation process (for example, if one wants to navigate only around the outside ring) or not (for example, if one wants to determine all edges of a face including all holes in this face). A similar idea is applied to a bridge face, but in this case a special attribute is assigned to a dual edge representing the bridge face. It is interesting to notice that a bridge edge in the primal graph represents a bridge face in the dual (and other way round – a primal bridge face represents a bridge edge in the dual).

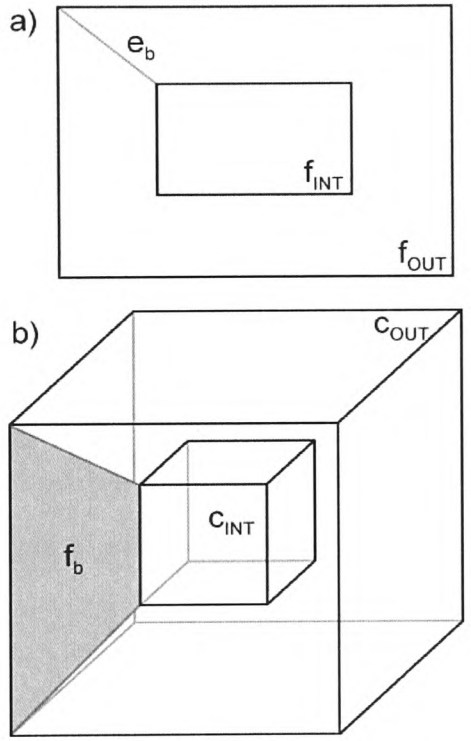

Figure 5.2 Holes and cavities: a) hole in a face – bridge edge e_b connects internal face loop f_{INT} with the outside loop f_{OUT}; b) cavity in a cell – bridge face f_b connects internal cell c_{INT} with the outside cell c_{OUT}.

5.5 Implementation details

5.5.1 Symbolism used

In this section implementation details are presented using pseudo-code. This is a blend of Object Pascal, C++ and descriptive language. Very often the code is simplified to emphasize its functionality and sometimes it can be expanded to make it useful in practice.

To make all examples easier to explain it was also assumed that: all faces are convex thus it is not possible that an edge belongs to one face – an edge is always a part of two face loops; all cells are convex thus it is not possible that faces of one cell are adjacent – two adjacent faces always belong to two different cells. These limitations do not affect the final functionality of the methods presented.

An edge is used as the basic construction element in the models that are B-Rep models. The idea of half-edges presented in Section 4.2 is used to split an edge into two directed halves that are represented with a symbol shown in Figure 5.3a): a half of an edge (straight line) is associated with a face (a square in the middle) and a vertex (a dot at the end). Such an element grouping a triple: a vertex, an edge, and a face (they are mutually incident) is called a *flag* (Grünbaum and Shephard, 1986). If the direction of a half-edge is important it is emphasized by an arrow at the end associated with the vertex (see Figure 5.3b).

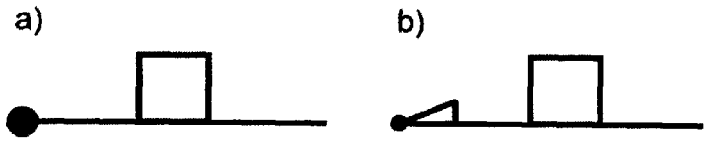

Figure 5.3 A half-edge symbol: a) a flag representation; b) the direction emphasized with an arrow.

5.5.2 Atomic element

All half-edges in a model are topologically connected using pointers. Two connected halves form an edge (half-edges *a* and *b* in Figure 5.4a). An edge divides two adjacent faces of a cell – an edge can be incident to one face if a face is not flat, for example a side face of a cylinder. All half-edges in a cell sharing the same vertex are connected and form a star (half-edges *a*, *b* and *c* in Figure 5.4b). However half-edges with the same vertex from different cells are not connected directly, but they share the same vertex. Half-edges bounding a face are also connected and form a loop (half-edges *a*, *b*, *c* and *d* in Figure 5.4c). Each half-edge in one space is permanently linked with the half-edge in the dual – this connection made in the construction process is never modified. This couple is called the *dual half-edge* (DHE). Half-edges in the dual are connected in the same way as in the primal.

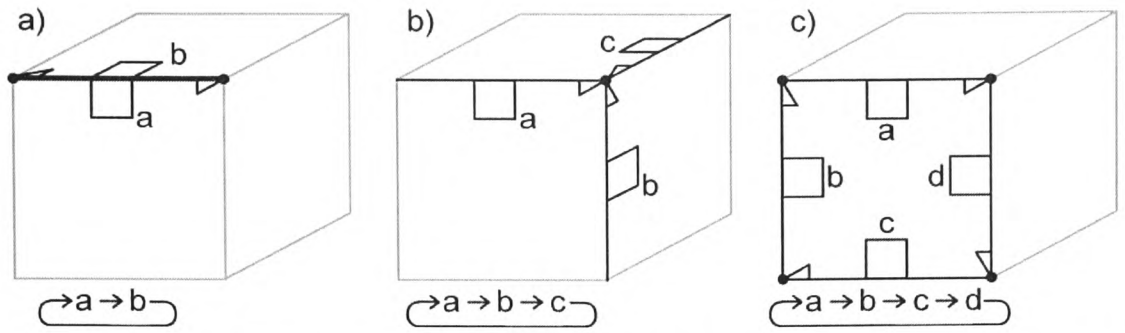

Figure 5.4 Half-edges in a cell are topologically connected and form: a) an edge; b) a star; c) a loop.

5.5.3 Pointer representation

A data structure used for the DHE representation consists of ten pointers: V, S, N_V, N_F, and D in each space – five in the primal and five in the dual. A reference to a vertex is assigned to the V pointer. Two half-edges are joined by the S pointer. N_V and N_F are used to store information about a next half-edge in a star (around a shared vertex) and in a loop (around a face) respectively. The next is considered as the next in anticlockwise direction looking from the outside of a cell. Half-edges from the primal and dual space are connected by the D pointer. These pointers are set during the construction process described later.

The basic classes are shown in Table 5.1. The *TDHE* class is the main class storing all the structure pointers. Two objects of the *TDHE* class represent a dual half-edge – one for primal space, and second for dual space. They are connected by the D pointer – the D pointer from one object points at the second one, and from the second object points at the first one. This connection is set in the construction process and is never changed, even if the model is modified. The *TVertex* class stores coordinates of a vertex in three-dimensional space. This class does not have any topological information included.

`class TVertex {`	`class TDHE {`
`x, y, z: Double;`	`V: TVertex;`
`}`	`S, NV, NF, D: TDHE;`
	`}`

Table 5.1 Basic classes used for DHE representation: *TVertex* is used for storing coordinates of vertices; dual half-edges are represented with *TDHE* – associated vertex and topological connections are stored as pointers.

5.5.4 Navigation

Navigation in a model is possible with a set of navigation operators that use the pointers described above. The navigation is performed from edge to edge (to be more precise – from half-edge to half-edge).

The basic set that uses pointers directly contains of four operators: *Sym*, $Next_V$, $Next_F$, and *Dual*. The *Sym* operator (see Figure 5.5a) uses the *S* pointer to navigate from one half of the edge to the second one. $Next_V$ (see Figure 5.5b) uses N_V and $Next_F$ (see Figure 5.5c) – N_F to navigate around a shared vertex and around a face respectively (in the anticlockwise direction looking from the outside of a cell). *Dual* (see Figure 5.40a) uses *D* to navigate from a half-edge in one space to the associated half-edge in the dual space.

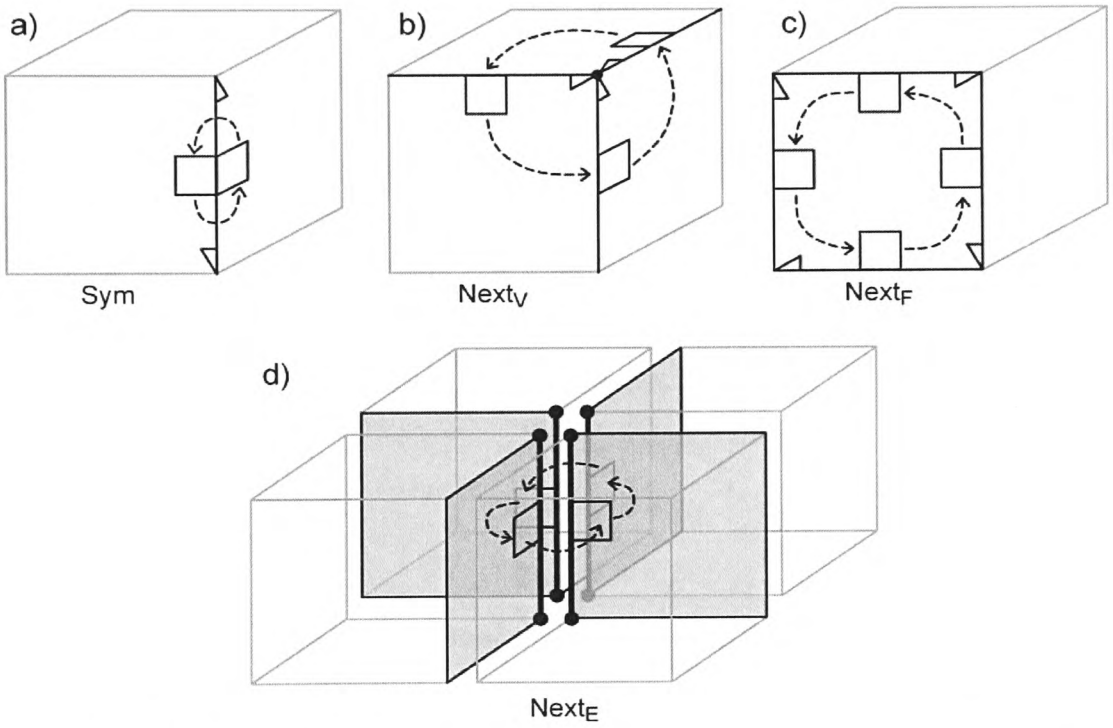

Figure 5.5 Navigational operators: a) *Sym* navigates from one half of an edge to the second one; b) $Next_V$ navigates around a shared vertex; c) $Next_F$ navigates around a face; d) $Next_E$ navigates around a bundle of edges.

Compound navigation operators based on the basic set are also defined: $Prev_V$, $Prev_F$, $Next_E$, $Prev_E$, and *Adjacent*. $Prev_V$ and $Prev_F$ allow for navigation in the same way as their counterparts $Next_V$ and $Next_F$ but in the opposite direction. $Next_E$ (see Figure 5.5d) and $Prev_E$ allow for navigation around a bundle of edges. *Adjacent* is used to navigate to the adjacent cell – the result of the operator is an edge in the adjacent cell that has the same coordinates and is associated with the opposite side of the shared face. In Figure 5.6a there are shown two adjacent cells (c_1 and c_2) that share a face. This face is double-sided – one side for each cell. The half-

edge e in c_1 shown in Figure 5.6b is one out of four edges forming the face loop. *e.Adjacent* is an adjacent half-edge in c_2.

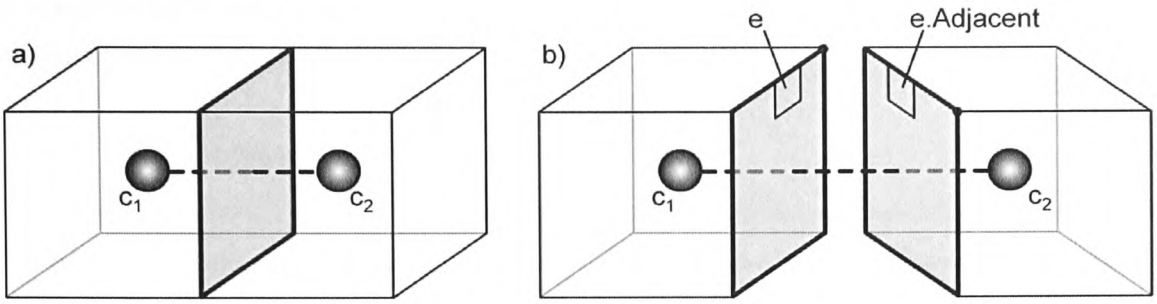

Figure 5.6 The *Adjacent* operator: a) the grey face is shared by adjacent cells c_1 and c_2; b) the *Adjacent* operator allows for navigation between cells: from e in c_1 to *e.Adjacent* in c_2

A pointer notation is used: thus, for example, the *Adjacent* operator is described by a sequence: $e.Adjacent = e.D.N_F.D.S$, where e is the source half-edge. This should be expanded as follows: go to the dual half-edge of e, then go the next counter-clockwise half-edge around a face, then go back to the original space, and go to the opposite side of the edge.

The full set of navigation operators is described by Equations 5.1 – 5.9. (Equivalent equations are given in parentheses.)

5.1 $$e.Sym = e.S$$

5.2 $$e.Next_V = e.N_V$$

5.3 $$e.Next_F = e.N_F$$

5.4 $$e.Dual = e.D$$

5.5 $$e.Prev_V = e.Dual.Next_V.Dual$$

5.6 $$e.Prev_F = e.Dual.Sym.Next_F.Dual.Sym$$

5.7 $$e.Adjacent = e.Dual.Next_F.Dual.Sym$$

5.8 $$e.Next_E = e.Dual.Next_F.Dual\ (e.Adjacent.Sym)$$

5.9 $$e.Prev_E = e.Sym.Dual.Next_F.Dual.Sym\ (e.Sym.Adjacent)$$

Using the full set of operators it is possible to develop more complex functions for a cell complex: for example finding all neighbours of a cell involves finding all the half-edges around a dual vertex representing the primal cell – the opposite ends of these dual edges point at vertices representing neighbouring cells.

Navigation between cells that are not directly connected is also possible, since a dual node represents a primal cell, and connections between cells are in the dual. Thus any graph traversal algorithm can be used to find a path between these two nodes. It starts from a half-edge

associated with the source node. Then a half-edge associated with the destination node is reached using simple pointer operators on the edges (nodes do not store topological connections). Consequently the path can be recorded as a sequence of topological connections starting from the source half-edge.

5.5.5 Construction

The construction of a 3D computer model represented as a cell complex has several stages. First the cells of a complex are created, and then all adjacent cells are connected. However, not all cells are the same: arbitrary polyhedra can have different shapes, a different number of faces, etc. Thus the process of cell construction should be 'atomized' to make incremental construction (edge by edge) possible (Boguslawski and Gold, 2010). This is possible with Euler operators, which are widely used in CAD for modifying B-Rep objects (see Section 4.4). They are 'atomic' operators that make only a minimal change in a model while preserving topological integrity (but do not provide rules to check or fix the topological consistency).

Construction of a single cell (without the dual) is a simple process using traditional Euler operators. For non-manifold models it is required to be able to construct more complex structures: to create non-manifold cell complexes the standard Euler operators should be extended to manage connections between cells and to include operations like joining two cells by a shared face, edge or vertex (Boguslawski and Gold, 2011).

Cells joined by a face is a normal situation in cell complex construction and does not need extra explanation. Given the external cell (see Section 5.4.1), two internal cells can be also joined by a shared edge or vertex. It is not possible to navigate directly from one cell to another; however these cells are connected via the external cell and navigation between the internal cells is possible in the dual. It should be noted that some non-manifold models can be simulated without the external cell by using the Cardboard & Tape method (see Section 5.6).

The construction operators are grouped in layers. Operators are dependent on operators from a lower level. Only operators from the lowest levels are based on pointers and other basic operations. Operators from higher levels are more complex and specialized. It is possible to build up new levels based on existing ones; however they would be application dependent. The highest level of the developed operators includes Euler operators. The full set is not covered, but a spanning set is implemented. Some extra Euler operators are developed to make the construction process easier and more intuitive. The developed set of Euler operators is shown in Table 5.2. They are described in detail below. Lower level functions called by these operators are presented later in this section.

MEVVFS/KEVVFS – Make/Kill Edge, Vertex, Vertex, Face and Shell
MEVFFS/KEVFFS – Make/Kill Edge, Vertex, Face, Face and Shell
MEV/KEV – Make/Kill Edge and Vertex
MVE/KVE – Make/Kill Vertex and Edge
MZEV/KZEV – Make/Kill Zero-length Edge and Vertex
MEF/KEF – Make/Kill Edge and Face
Join/Separate by Face/Edge/Vertex (Extended Euler Operators)
Merge/Split by Face/Edge/Vertex (Extended Euler Operators)

Table 5.2 A set of Euler operators.

Make/Kill Edge, Vertex, Vertex, Face and Shell – MEVVFS / KEVVFS

Except for *MEVFFS*, *MEVVFS* is the only operator that can be used to create a new cell (shell). Basically a new edge in empty space is created. This edge forms a cell that may be further developed to obtain a polyhedron. However this process is more complex because the external cell and dual edges are also constructed. There are four edges involved in this process – two in the primal, and two in the dual (see Figure 5.7).

There are four input parameters: two vertices in primal space (*P1* and *P2*), and two in the dual (*I* and *E*) (see Table 5.3). Vertices *P1* and *P2* bound a new edge. Vertices *I* and *E* are dual nodes representing the internal and external cells of a new complex. In this case these cells are formed by dangling edges: the edge built of *a* and *b* half-edges forms an internal cell associated with the *I* vertex; the edge built of *c* and *d* half-edges forms an external cell associated with the *E* vertex. All connections set by the operator are shown in Table 5.4: half-edges created in the process ($a - h$) are shown in the first column, and the value of pointers (S, N_V, N_F, D and V) in the rest of the columns.

MEVVFS is not a 'perfect' Euler operator because it introduces many elements into a model: one edge, two vertices, one face and one shell. The reason is that: an edge is a minimal topological and valid element allowing for navigation; an isolated vertex does not bring any topological information thus cannot be created without an edge and placed in empty space; a face is created automatically and is a result of the DHE representation; the number of shells is determined by the number of cell complexes mutually disconnected.

KEVVFS is the reverse operator that removes a dangling edge (not connected to any other edge), and in the result the space is left empty. Because this operator reduces the number of shells by one, the removed edge has to be the only element of a shell and other edges cannot be connected to the edge. Before the edge is removed a test should be performed to check this condition.

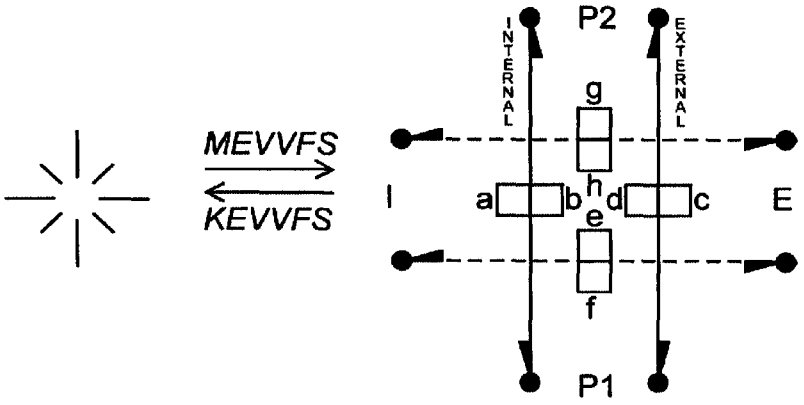

Figure 5.7 *MEVVFS* and *KEVVFS* operators.

```
function MEVVFS(P1, P2, I, E) {
Result:=MakeComplexEdge(P1, P2, I, E);
}
```

```
function KEVVFS(e) {
KillComplexEdge(e);
}
```

Table 5.3 *MEVVFS/KEVVFS* operators.

	S	N_V	N_F	D	V
a	b	a	b	e	P1
b	a	b	a	g	P2
c	d	c	d	f	P1
d	c	d	c	h	P2
e	f	e	f	a	I
f	e	f	e	c	E
g	h	g	h	b	I
h	g	h	g	d	E

Table 5.4 Table of connections made by *MEVVFS*.

Make/Kill Edge, Vertex, Face, Face and Shell – MEVFFS / KEVFFS

MEVFFS is very similar to *MEVVFS*. However this operator creates a degenerate edge – an edge bounded by the same vertex. It can be imagined as a curved edge forming a loop, or as a zero length edge. In Figure 5.8 this edge is represented as a loop.

There is one input parameter less than in *MEVVFS* because two vertices from primal space (*P1* and *P2*) are replaced by one – *P* (see Table 5.5). The connections set by this operator are shown in Table 5.6.

The reverse operator – *KEVFFS* – works in the same way as *KEVVFS* – it removes a dangling edge and reduces a number of shells by one. Before the edge is removed it is necessary to check if the edge is the only element forming the shell (the edge cannot be connected to any other edge).

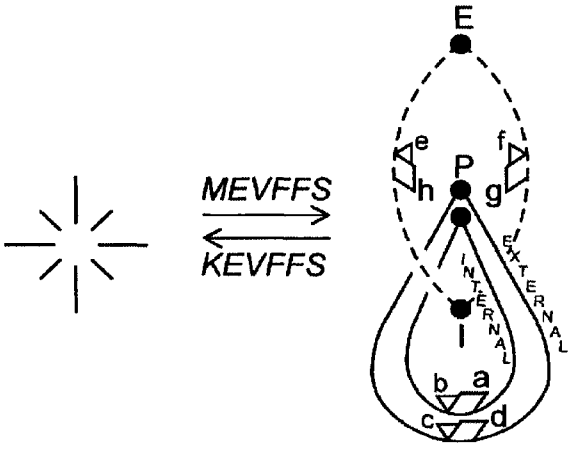

Figure 5.8 *MEVFFS* and *KEVFFS* operators.

```
function MEVFFS(P, I, E) {
e:=MakeComplexEdge(P, P, I, E);
ComplexSplice(e, e.Sym);
Result:=e;
}
```

```
function KEVFFS(e) {
KillComplexEdge(e);
}
```

Table 5.5 *MEVFFS/KEVFFS* operators.

	S	**N_V**	**N_F**	**D**	**V**
a	b	b	a	e	P
b	a	a	b	g	P
c	d	d	c	f	P
d	c	c	d	h	P
e	f	g	f	a	I
f	e	h	e	c	E
g	h	e	h	b	I
h	g	f	g	d	E

Table 5.6 Table of connections made by *MEVFFS*.

Make/Kill Edge and Vertex – MEV / KEV

MEV is the first operator out of three that create an edge and a vertex. Two other operators: *MVE* and *MZEV* described later add the same entities to a model but create different connections and their interpretation is different. They could be called by the same name (*MEV*) but to distinguish their functionality there are three separate operators: *MEV*, *MVE* and *MZEV*. This ambiguity of the *MEV* operator is mentioned in Section 4.4.

MEV creates a vertex and an edge (see Figure 5.9): the new edge e_N is linked to an existing model in such a way that one end of the edges is free (not connected to any other edge). The new vertex *P* bounds the edge at one end; the second end is bounded by a vertex already existing at the point of connection with a model determined by the half-edge *e*.

It is important to notice that MEV modifies two cells at once: the original one, and the cell adjacent to the original, where the adjacency by a face represented by the input half-edge *e* is considered. The original cell is modified explicitly – a new edge is added after *e* in a disc cycle around *e.V*, and before *e* in a face loop determined by *e*. An edge is also added to the adjacent face (*e.Dual.Sym.Dual*) because in the model adjacent faces must be identical. This process is represented by two functions (see Table 5.7): *MakeComplexEdge* which constructs two edges (one for the original cell and one for the adjacent cell), and *ComplexSplice* which links edges to the right cells.

Dual edges are automatically created and connected with the existing model. Dual vertices necessary to create new dual edges are taken from the existing model (*e.D.V* and *e.Adjacent.D.V* – see Table 5.7). Dual edges and adjacent cells are not shown in Figure 5.9.

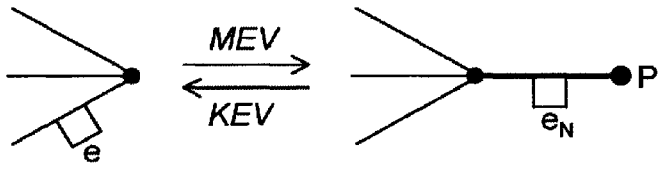

Figure 5.9 *MEV* and *KEV* operators.

```
function MEV(P, e) {
eN:=MakeComplexEdge(e.V, P, e.Dual.V, e.Adjacent.Dual.V);
ComplexSplice(eN, e);
Result:=eN;
}
```

```
function KEV(e) {
ComplexSplice(e, e.Sym.NextF);
KillComplexEdge(e);
}
```

Table 5.7 *MEV/KEV* operators.

KEV unlinks an edge from the model, and then removes the edge. Before the edge is removed, it should be checked to see if the second end of the edge is free (not connected to any other edge).

Make/Kill Vertex and Edge – MVE / KVE

MVE is similar to the *MEV* operator – entities created by the operators are the same: an edge and a vertex. The difference between these operators is as follows: *MEV* is used to add a new edge to the model, and one end of the edge stays free; *MVE* splits an existing edge into two parts divided by a new vertex (see Figure 5.10). At first glance *MVE* should not differ much from *MEV*, but there is one significant issue that has to be taken into consideration: the proposed Euler operators work for cell complexes, therefore the *MVE* operator can be used on an edge in a bundle of edges (the number of edges depends on number of cells sharing the bundle). That means the *MVE* operator creates as many new edges as those that already exists in a bundle – one edge for each cell sharing the bundle (see Figure 5.11). This makes the code more complicated (see Table 5.8).

Dual edges created by the *MVE* operator form a flat cell around the newly created primal vertex. There is no need to 'organize' dual connections in the reverse *KVE* operator because all the dual edges forming the dual cell are removed – the flat cell is removed from between two adjacent cells and all the connections between these two cells are set automatically.

KVE merges two edges into one and removes a vertex. An input parameter *e* (see Table 5.8) is one of the edges in a bundle that is directed from the point to remove – half-edge *c* or *d* in Figure 5.10.

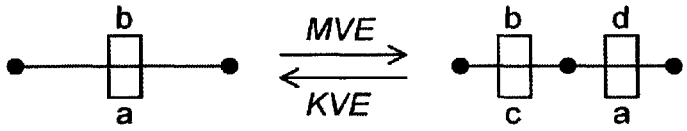

Figure 5.10 *MVE* and *KVE* operators.

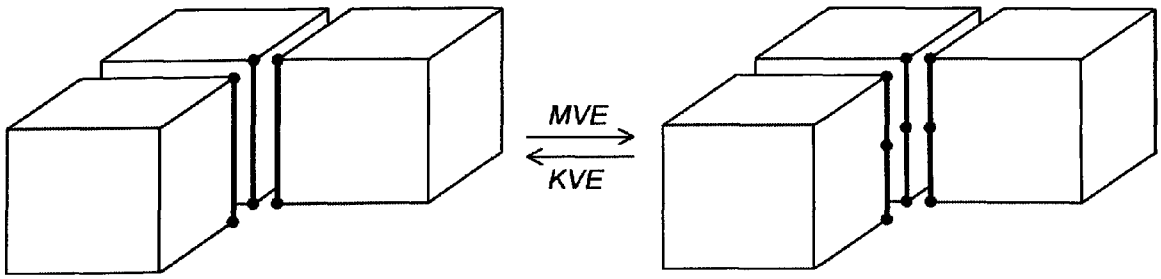

Figure 5.11 *MVE* works for cell complexes – all edges in a bundle shared by many cells are split.

```
function MVE(P, e) {
for eB:=all edges in a bundle e
   newEdgesList.Add(MakeDegenerateEdge(P, eB.Dual.V));
i:=0;
for eB:=all edges in a bundle e {
   eN:=newEdgesList(i);
   eNN:=newEdgesList((i+1) mod newEdgesList.Count);
   Snap(eN, eB);
   Snap(eN.Dual.Sym, eNN.NextV.Dual);
   i++;
   }
Result:=newEdgesList(0);
}
```

```
function KVE(e) {
for eB:=all edges in a bundle e
   Snap(eB.NextV.Sym, eB);
   KillDegenerateEdge(eB);
}
```

Table 5.8 *MVE/KVE* operators.

Make/Kill Zero-length Edge and Vertex – MZEV / KZEV

This is the last operator in the collection that creates an edge and a vertex. The idea of *MZEV* is to split a vertex (or duplicate a vertex) and to add an edge in between. There are two input parameters: *e1* and *e2* (see Table 5.9) – they are half-edges that determine the place to insert a new edge. To understand this situation (see Figure 5.12), imagine several edges forming a disc cycle around a shared vertex. In order to split the vertex it is necessary to divide the cycle into two parts. These two parts (a place to divide the cycle) are determined by two edges from a cycle. The idea of the disc cycle division is similar to the quad-edge Splice operator (Guibas and Stolfi, 1985).

The use of the *MZEV* operator is limited to a single cell (a single internal cell and its external cell), and does not work for cells in a complex: it is not possible to automatically and unambiguously determine all the neighbouring cells that should be modified to preserve correct connections.

KZEV removes the edge *e1* from between two vertices and one vertex *e1.Sym.V* (see Table 5.9).

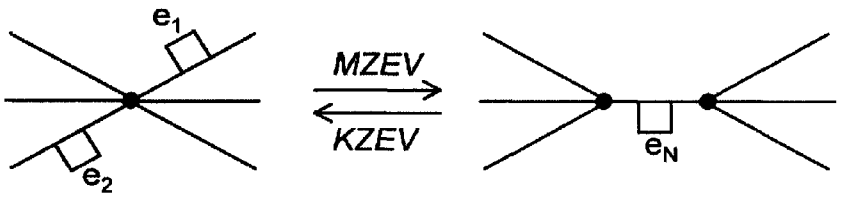

Figure 5.12 *MZEV* and *KZEV* operators.

```
function MZEV(e1, e2) {
Vn:=duplicate the e1.V vertex;
ComplexSplice(e1, e2);
Result:=MEF(e1, e2);
for e:=all edges around e2.V
   setVertex(e, Vn);
   setVertex(e.Adjacent.Sym, Vn);
}
```

```
function KZEV(e) {
for eN:=all edges around e.Sym.V
   setVertex(eN, e.V);
   setVertex(eN.Adjacent.Sym, e.V);
he1:=e.PrevV;
he2:=e.Sym.PrevV;
KEF(e);
ComplexSplice(he1, he2);
}
```

Table 5.9 *MZEV/KZEV* operators.

Make/Kill Edge and Face – MEF / KEF

MEF splits a face loop into two parts by adding an edge (see Figure 5.13). Because a face to be split is shared by two cells, an adjacent face is also split and a new edge is added. The operator is simple: a new edge is added between two input edges *e1* and *e2* (each end of the new edge is put into a disc cycle determined by *e1* and *e2* – a face loop is modified automatically) (see Table 5.10). It is important to find if these edges are part of one face loop. If not it is necessary to deal with other cases: *MEKH* (Make Edge Kill Hole) – if *e1* and *e2* belong to two different face loops but one loop is enclosed by the second one (hole), and *MEKFS* (Make Edge Kill Face and Shell) if *e1* and *e2* belong to two separate cells. These operators do the same and together with *MEF* form a collection which adds a new edge in between two vertices. However their meanings are different.

KEF – the reverse operator – removes an edge and merges two face loops into one loop.

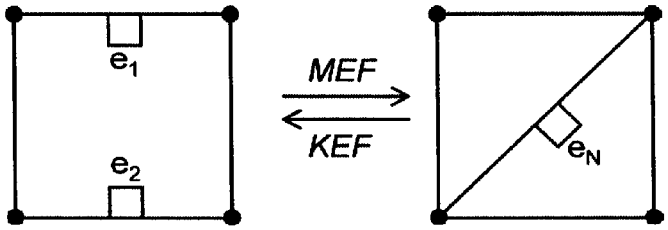

Figure 5.13 *MEF* and *KEF* operators.

```
function MEF(e1, e2) {
e:=MakeComplexEdge(e1. V, e2. V,
       e1.Dual.V, e1.Adjacent.Dual.V);
ComplexSplice(e, e1);
ComplexSplice(e2, e.Sym);
Result:=e;
}
```

```
function KEF(e) {
eSym:=e.Sym;
eNextF:=e.NextF;
ComplexSplice(e, eSym.NextF);
ComplexSplice(eSym, eNextF);
KillComplexEdge(e);
}
```

Table 5.10 *MEF/KEF* operators.

Join/Separate

Join/Separate and *Merge/Split* belong to a set of extended Euler operators described in Section 4.4.

'*Join*' is a collection of three operators to connect two cells – it is possible to join cells by a common face, edge or vertex (see Figure 5.14). The relationships between cells are changed in this way so that direct navigation between the cells is possible and the cells are part of the same complex even if they were in two different complexes before the operation. The cells are not modified themselves – only the connections between them and external cells are modified.

Cell joining by a face is the most useful operator for cell complex construction. In building interior modelling (the main objective in this research) escape routes and navigation between rooms is an important issue. Rooms are represented by cells. Only navigation from cell to cell through faces (doors, windows, walls, etc.) is permitted. Therefore cells are not joined by a common edge or vertex even if they geometrically fit and such a connection is possible. However the full set of Join operators may be important in other applications.

'*Separate*' is the reverse of *Join*. This operator disconnects two cells, thus direct navigation between them is not possible. If there is no alternative path from one cell to the second one it means that the cells are in different cell complexes after separation.

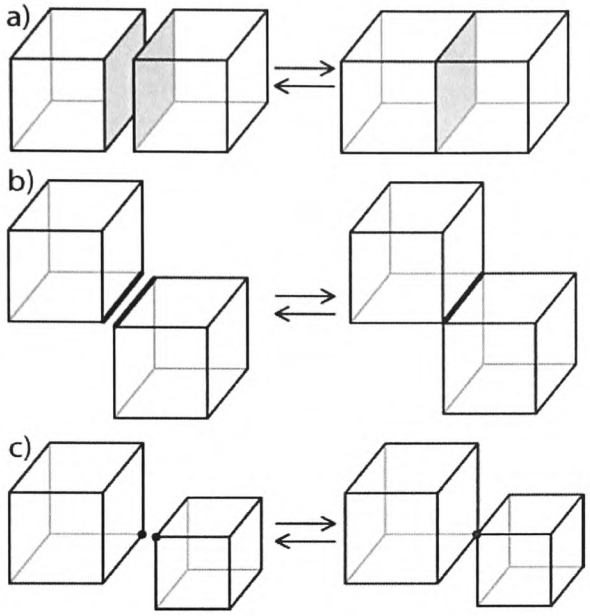

Figure 5.14 *Join/Separate* by a common a) face; b) edge; c) vertex.

'*Join by Face*' joins two cells by a common face (see Figure 5.15). This connection is possible only if the faces have the same number of edges. They do not have to have the same vertices. However visualization may give strange results if the faces to be connected do not fit geometrically.

Assume one wants to join two cubes by a face. Each of them is enclosed by an external cube – there are four cells in two cell complexes. Both of them have one identical face – this will be the common (shared) face after the connection. The edges of the internal cubes are not changed – only the identical faces of the external cubes are removed: external cubes are merged into one external cell (see Figure 5.15a). Dual connections are changed automatically so that direct navigation between the internal cubes is possible. The result of the operation is one cell complex made of three cells – two internal cubes and one external cell. The pseudo-code for this function is shown in Table 5.11 – the 'corresponding edge' term needs a short explanation: two edges correspond one to another if after joining they are in the Adjacent relationship and usually they have the same bounding vertices. Input parameters *e1* and *e2* represent edges from internal cells which will be in the Adjacent relationship after joining (see Figure 5.15b).

'*Separate by Face*' is the reverse operator that disconnects two cells.

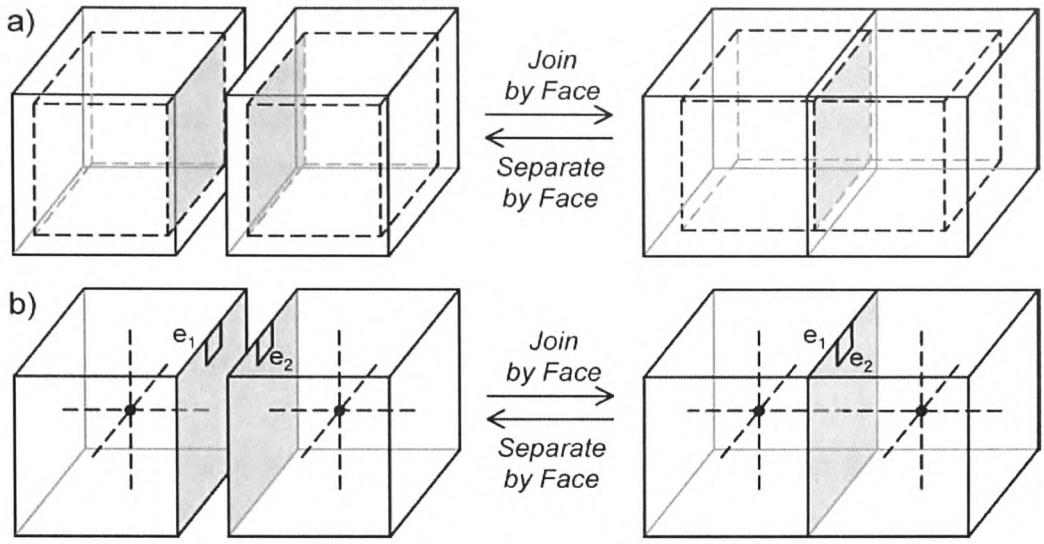

Figure 5.15 *Join/Separate by Face* operators: a) internal cells (dashed lines) are joined by a shared face (grey face); external cells (solid lines) are merged; b) internal cells with the dual (dashed lines); e_1 and e_2 represent adjacent faces.

```
function JoinByFace(e1, e2) {
faceToKill:=e1.Adjacent;
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face e2
  if f1.Adjacent<>f1.Adjacent.Sym
     Sew(f1.Adjacent, f2.Adjacent.Sym);
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face e2
  Snap(f1.NextF.Adjacent.Dual.Sym, f2.Adjacent.Dual);
KillFace(faceToKill);
}
```

```
function SeparateByFace(e1, e2) {
//e2 - an adjacent face to e1 (e1.Adjacent)
newFace:=MakeFace(list of primal vertices of a face loop e1,
                  dual vertex in infinity);
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face newFace
  Snap(f1.Dual.Sym, f2.Dual);
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face e2
  Sew(f1.Adjacent, f2. Adjacent.Sym);
}
```

Table 5.11 *Join/Separate by Face* operators.

'*Join by Edge*' is the second operator from the *Join* collection. The implementation is much simpler than *Join by Face* (see Table 5.11). The common edges from the external cells are combined (see Figure 5.16a) and nothing is removed from the model. The input parameters and half-edges e_1 and e_2 (see Table 5.12), are shown in Figure 5.16b. After the join, the dual edges associated with the shared edges form a face delimited by the dual vertices representing linked cells and the external vertex V_E (representing the external cell) – there is only one external vertex, but it was shown as two vertices to make the picture clearer. Thus navigation around the shared edge (in fact this is a bundle of edges) is possible – the navigation is performed around the dual face.

'*Separate by Edge*' is the reverse operator. The code of the function is the same as *Join by Edge* because it uses the self-reversible *Sew*.

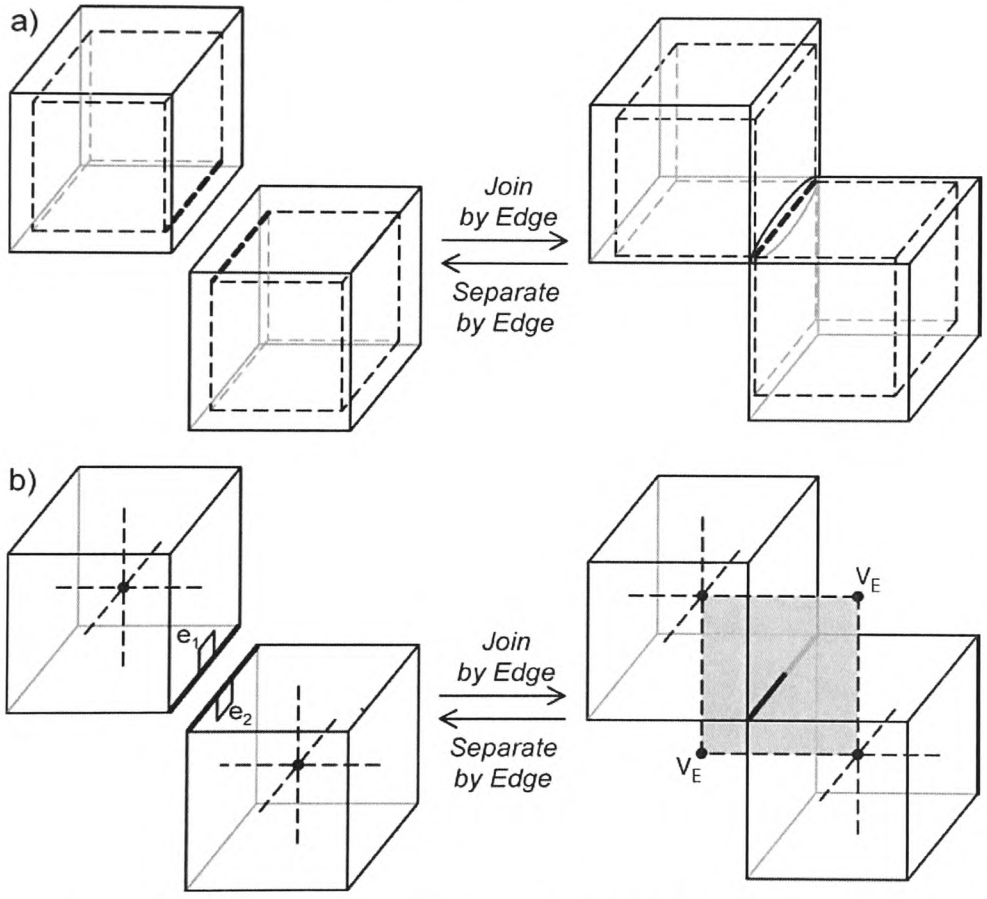

Figure 5.16 *Join/Separate by Edge* operators: a) internal cells (dashed lines) are joined by a shared edge (bold dashed edge): external cells (solid lines) are merged; b) internal cells with their dual (dashed lines): e_1 and e_2 represent adjacent edges; the grey dual face is penetrated by the shared edge (a bundle of edges); V_E – the external vertex.

```
function JoinByEdge(e1, e2) {
Sew(e1.Adjacent, e2.Adjacent);
}
```

```
function SeparateByEdge(e1, e2) {
Sew(e1.Adjacent, e2.Adjacent);
}
```

Table 5.12 *Join/Separate by Edge* operators.

'*Join by Vertex*' is the last operator from the *Join* collection. It merges the external cells like the two previous operators (see Figure 5.17a): numbers in circles (1 – 6) determine an edge's order around the shared vertex in the external cell after the joining in the case of choosing edges e_1 and e_2 (see Figure 5.17b) as input parameters (see Table 5.13). These edges determine the connection point of two disc cycles in external cells (vertices have no topological information thus cannot be used to define this point). The result is similar to the quad-edge Splice that merges two disc cycles but also dual connections need to be managed. There is an exception made for *Splice* in the joining by vertex operation – the N_F pointers are not changed. Otherwise the two face loops represented by *e1* and *e2* would be merged into one loop which is not the correct result. The dual edges associated with the primal edges around the shared vertex form a cell enclosing the primal shared vertex (the grey cell in Figure 5.17b) – the two corners are determined by the dual vertices of the cells engaged in the join operation: the rest of the corners are determined by the external vertex (there is only one external vertex, but it was presented as the remaining six vertices of the grey box to make the picture clearer).

'*Separate by Vertex*' is the reverse operator.

```
function JoinByVertex(e1, e2) {
Splice(e1.Adjacent.Sym.PrevV, e2.Adjacent.Sym.PrevV,
          do not change NF pointers);
}
```

```
function SeparateByVertex(e1, e2) {
Splice(e1.Adjacent.Sym.PrevV, e2.Adjacent.Sym.PrevV,
          do not change NF pointers);
}
```

Table 5.13 *Join/Separate by Vertex* operators.

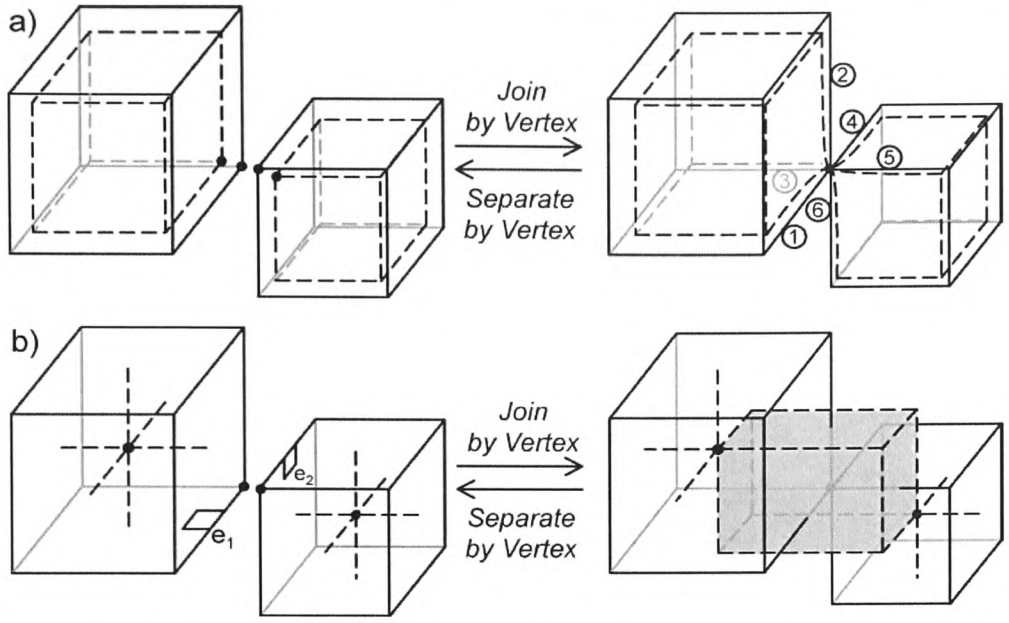

Figure 5.17 *Join/Separate by Vertex* operators: a) internal cells (dashed lines) are joined by a shared vertex: external cells (solid lines) are merged; b) internal cells with their dual (dashed lines): e_1 and e_2 represent two edges from two disc cycles; the grey dual cell encloses the shared primal vertex.

Merge/Split

'*Merge*', like the *Join* operator, is a collection of three operators – it is possible to merge cells by a common face, edge or vertex (see Figure 5.18). In this case two cells are merged into one cell. The main difference to *Join* is that all changes directly affect the internal cells – not external cells. Cells to be connected have to be joined first. Merging two cells that are in separate complexes has two stages: first cells are joined, then they can be merged. However it is possible to develop a new operator to join and merge cells in one step.

'*Merge by Face*' allows for merging two cells joined by a shared face into one cell – the shared face is removed from between the cells and then deleted (see Figure 5.19). This operation changes only the internal cells, thus it can be used inside a cell complex where the external cell is not directly accessible.

The code (see Table 5.14) is almost the same as *Join by Face* (see Table 5.11) with one difference – in the merge version the internal cells represented by *e1* and *e2* are modified while in the join version external cells are changed: *e1.Adjacent* and *e2.Adjacent*.

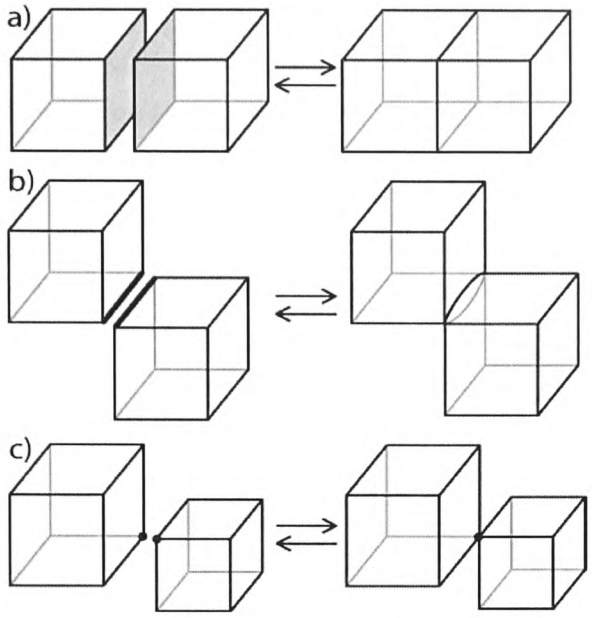

Figure 5.18 *Merge/Split* by a common a) face; b) edge; c) vertex.

'*Split by Face*' is the reverse operator to split one cell into two parts. This is more complicated than its join equivalent. The reason is that there is no information about a face which 'cuts' a cell into two parts and then is inserted between these parts. Information to construct such a face is obtained from a list of edges given as a parameter (see Table 5.14). Edges on the list should fulfil some rules: edges form a cycle – an end of one edge should be the beginning of the following edge on the list, where the last item on the list is followed by the first one (a cyclic list); all edges lie on the same plane – only flat faces in the model are assumed; all edges on the list exist in the model and they are part of the cells to be merged.

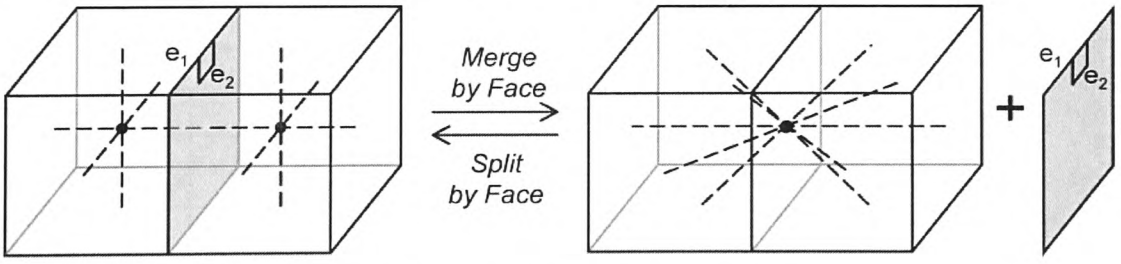

Figure 5.19 *Merge/Split by Face* operators.

```
function MergeByFace(e1, e2) {
faceToKill:=e1;
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face e2
  if f1<>f2.Sym
    Sew(f1, f2.Sym);
for f1:=all edges from face e1 and
      f2:=all corresponding edges from face e2
  Snap(f1.NextF.Dual.Sym, f2.Dual);
KillFace(faceToKill);
}
```

```
function SplitByFace(eList) {
for e:=all edges from eList
  pList.Add(e.V);
newFace:=MakeFace(pList, eList(0).Dual.V);
for e:=all edges from eList
  Sew(e, newFace);
  newFace:=newFace.NextV.Sym;
}
```

Table 5.14 *Merge/Split by Face* operators.

'*Merge by Edge*' is used to combine two cells sharing an edge (see Figure 5.20). The code of this function is very similar to *Join by Edge* – only the input parameters for *Sew* are different (see Table 5.15) because this time the internal cells are changed – not the external cells.

'*Split by Face*' is the reverse operator. After the split the cells are still in the same complex.

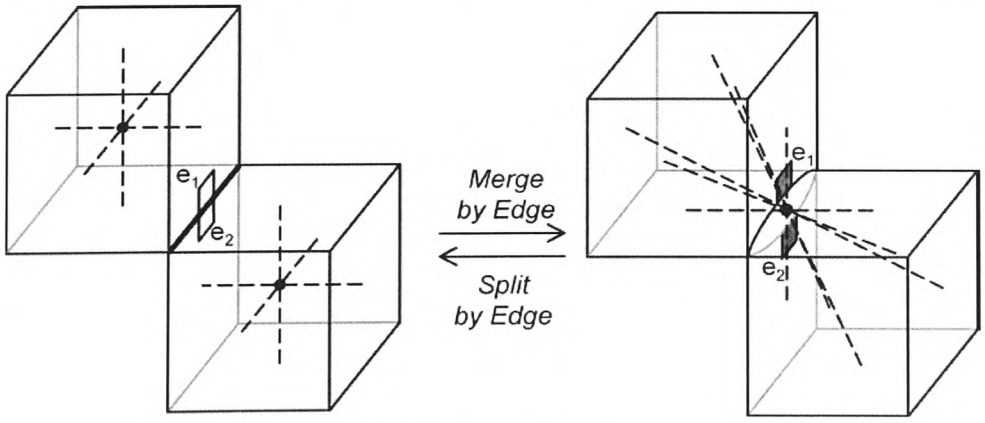

Figure 5.20 *Merge/Split by Edge* operators.

```
function MergeByEdge(e1, e2) {
Sew(e1, e2);
}
```

```
function SplitByEdge(e1, e2) {
Sew(e1, e2);
}
```

Table 5.15 *Merge/Split by Edge* operators.

'*Merge by Vertex*' is used to connect two cells by a shared vertex (see Figure 5.21). In this case the function is similar to the *Join* equivalent – *Splice* is performed with different input parameters (see Table 5.16).

'*Split by Vertex*' is the reverse operator. After the split the cells are still in the same complex.

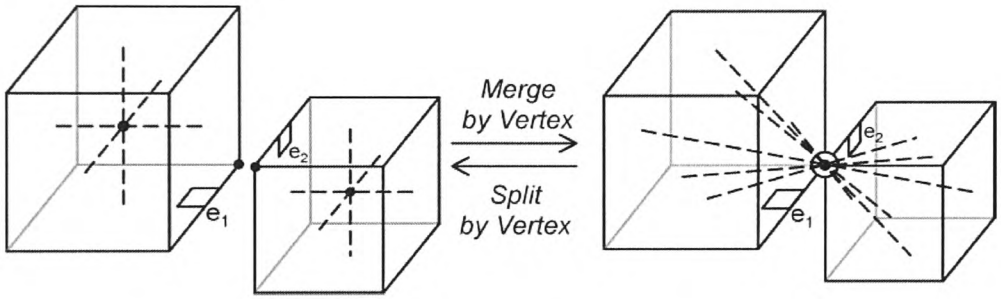

Figure 5.21 *Merge/Split by Vertex.*

```
function MergeByVertex(e1, e2) {
Splice(e1, e2, do not change NF pointers);
}
```

```
function SplitByVertex(e1, e2) {
Splice(e1, e2, do not change NF pointers);
}
```

Table 5.16 *Merge/Split by Vertex* operators.

The operators described so far are included in the highest level of the developed operators. They should be used with care because no conformity with the geometry is checked. For example, *Join by Face* joins two cells by adjacent faces but it is not tested if the two input parameters represent adjacent faces. If these faces are spread out in a model, joining them may give strange results, especially in visualization. However it is possible to develop higher level of operators for testing the geometrical relationships before the right operator is performed.

The presented Euler operators do not modify the topological relations directly – they use lower level operators presented below (see Table 5.17). These are presented below together with the operators from the lowest layers which directly change the pointers.

Make / Kill Complex Edge
Make / Kill Face
Sew
Complex Splice
Splice
Half-Splice
Snap
Make / Kill Edge
Make / Kill Degenerate Edge
Make / Kill Half-Edge
Connect / Disconnect Half-Edges

Table 5.17 Lower level operators used by Euler operators.

Make / Kill Complex Edge

This function is used to create a new complex edge – in fact there are four edges created at once: one internal, one external, and two dual edges. All these edges are linked, thus navigation between them is possible. This is the reason why the new edge is called complex.

The *MEVVFS* operator performs only *Make Complex Edge* and no other functions are called (see Table 5.18 and Figure 5.7). *Kill Complex Edge* is also the only function performed by *KEVVFS* – this removes a complex edge. The edge to be removed cannot be connected to any other edge.

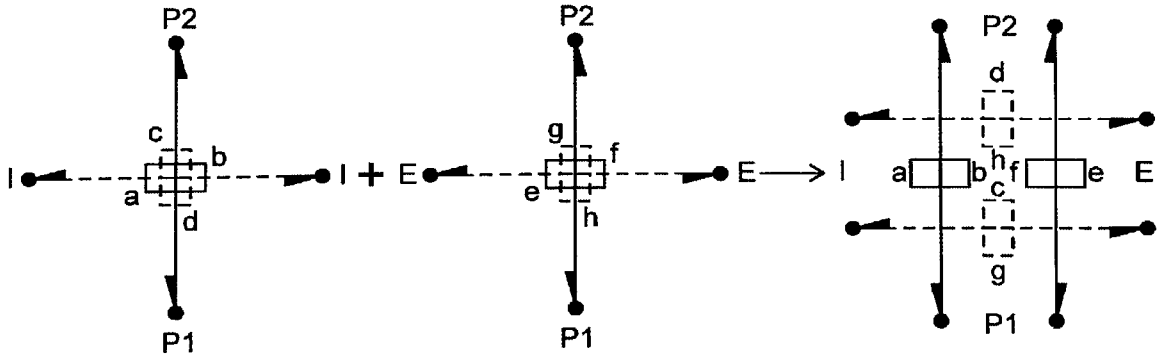

Figure 5.22 *Make Complex Edge* snaps two edges in dual space.

```
function MakeComplexEdge(P1, P2, I, E) {
eINT:=MakeEdge(P1, P2, I);
eEXT:=MakeEdge(P1, P2, E);
Snap(eINT.Dual.Sym, eEXT.Dual);
}
```

```
function KillComplexEdge(e) {
KillEdge(e.Adjacent);
KillEdge(e);
}
```

Table 5.18 *Make/Kill Complex Edge* operators.

Make / Kill Face

Make Face is used to construct a double-sided face. The input parameters include a list of face vertices (*pList:* P_0, P_1, ...) and a dual vertex (*D*) (see Table 5.19): there is only one dual vertex, however dual edges created by the operator are not connected in a disc cycle around this vertex (but it is possible to navigate between them using primal space). First a degenerate edge (bounded by one vertex) is created. This is the simplest double-sided face possible in the model. The newly created edge consists of half-edges a_0 and b_0 which are associated with the vertex P_0 and dual half-edges c_0 and d_0 respectively (see Figure 5.23). The dual edge forms a 'balloon' around the primal vertex. Then the next degenerate edge associated with the next vertex P_1 is created and added to the face using *Snap* described below – this edge consists of half-edges a_1 and b_1 and their dual counterparts c_1 and d_1. This process is iteratively repeated until all vertices from the list are used. *Make Face* is used by *Split by Face*.

Kill Face removes all edges of a face – this face must be disconnected from a model beforehand.

```
function MakeFace(pList, D) {
e1:=MakeDegenerateEdge(pList(0), D);
for P:=all vertices from pList starting from the second element
  e2:=MakeDegenerateEdge(P, D);
  Snap(e1.NextV, e2);
Result:=e1;
}
```

```
function KillFace(e) {
for e:=all edges from the face loop e
  KillEdge(e);
}
```

Table 5.19 *Make/Kill Face* operators.

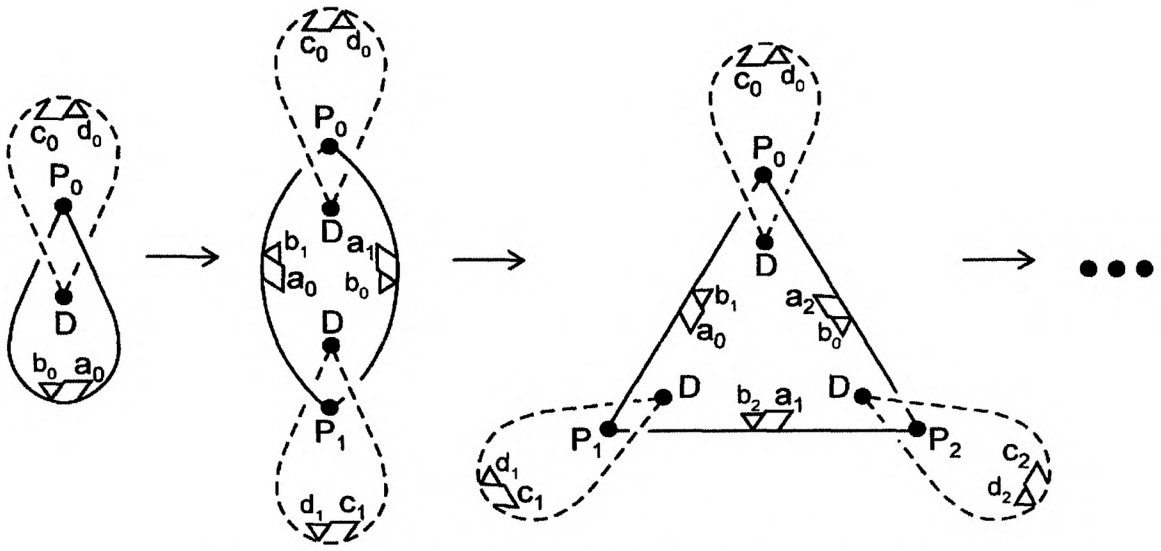

Figure 5.23 Process of face construction in the *Make Face* operator.

Sew

Sew is a self-reversible operator (see Table 5.20) to combine two double-sided faces as presented in Figure 5.24.

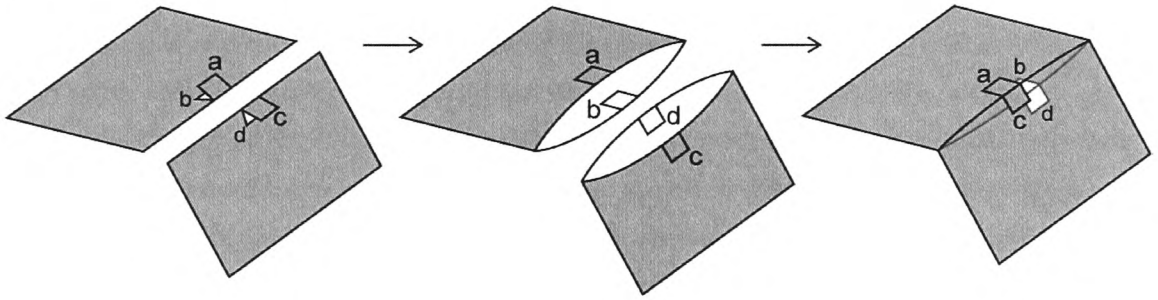

Figure 5.24 A process of connecting two faces in the *Sew* operator.

```
function Sew(e1, e2Sym) {
e1Sym:=e1.Sym; e2:=e2Sym.Sym;
e1PrevV:=e1.PrevV; e2PrevV:=e2.PrevV;
e1SymPrevV:=e1Sym.PrevV; e2SymPrevV:=e2Sym.PrevV;

Snap(he1, he2Sym);
Splice(he1PrevV, he2SymPrevV);
Splice(he1SymPrevV, he2PrevV);
}
```

Table 5.20 The *Sew* operator.

Complex Splice

Complex Splice is another self-reversible operator (see Table 5.21). This is used to join an edge to the rest of the model in the process of cell construction. An edge in the external cell and the dual edges are also joined with the existing structure. The result is similar to the quad-edge Splice but the complex version manipulates complex edges. The idea of the simple *Splice* is presented below.

```
function ComplexSplice(e1, e2) {
e1Temp:=e1.Dual.Sym.Dual;
e2Temp:=e2.Dual.Sym.Dual;

Splice(e1, e2);
Splice(e1Temp, e2Temp);
Snap(e1.Dual.Sym, e2Temp.Dual);
Snap(e2.Dual.Sym, e1Temp.Dual);
}
```

Table 5.21 The *Complex Splice* operator.

Splice and Half Splice

Splice is a self-reversible operator working on single edges – however the primal and the linked dual edges are considered. *Splice* is split into two parts – two *Half Splices* are performed: one for the dual edge and the second for the primal edge (see Table 5.22). The *Half Splice* operator is similar to the quad-edge Splice – two disc cycles are merged into one (N_V pointers in half-edges e_1 and e_2 are changed) and two face loops are modified (the original faces f_1 and f_2 are split into a different configuration f_I and f_{II}) (see Figure 5.25).

The *NF_Change* input parameter (see Table 5.22) is only important for *Join/Separate* and *Merge/Split by Vertex*. In the case of these operators the face loops are not changed. In all other cases (a default value *NF_Change=true*) the face loops are modified.

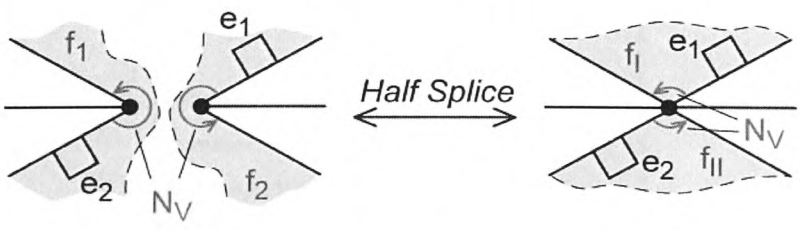

Figure 5.25 The *Half Splice* operator.

```
function Splice(e1, e2, NF_Change=True) {
HalfSplice(e1.NextV.Dual, e2.NextV.Dual, NF_Change);
HalfSplice(e1, e2, NF_Change);
}
```

```
function HalfSplice(e1, e2, NF_Change=True) {
e1Next:=he1.NextV; e2Next:=he2.NextV;

e2.NV:=e1Next;
e1.NV:=e2Next;

if NF_Change
  e1Next.Sym.NF:=e2;
  e2Next.Sym.NF:=e1;
}
```

Table 5.22 *Splice* and *Half Splice* operators.

Snap

Snap is used to merge the face loops and exchange the half-edges of two edges (see Figure 5.26): half-edges of the input edges e_1 and e_2 (see Table 5.23) are exchanged and N_F pointers are changed. Thus the top sides (f_1 and f_3) and the bottom sides (f_2 and f_4) of the faces are merged. *Snap* is a self-reversible operator that works in one space only (primal or dual).

```
function Snap(e1, e2) {
eTemp:=e1.Sym.NextF;
e1.Sym.NF:=e2.Sym.NextF;
e2.Sym.NF:=eTemp;

eTemp:=e1.NextF;
e1.NF:=e2.NextF;
e2.NF:=eTemp;

eTemp:=e1.Sym;
ConnectHalfEdges(e1, e2.Sym);
ConnectHalfEdges(e2, eTemp);
}
```

Table 5.23 The *Snap* operator.

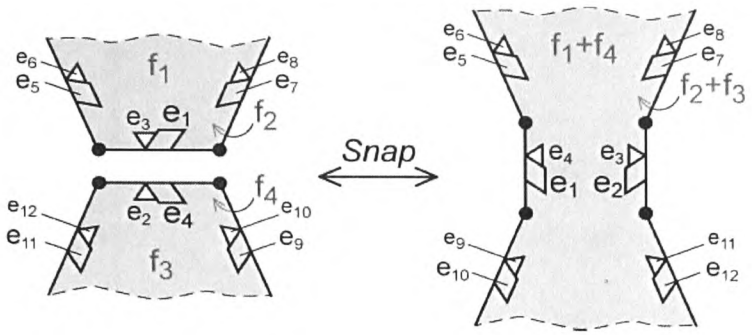

Figure 5.26 The *Snap* operator.

Connect / Disconnect Half-Edges

Connect Half-Edges links two half-edges by the *S* pointer into a single edge (see Table 5.24 and Figure 5.27). This operator works in one space only (primal or dual).

Disconnect Half-Edges is the reverse operator that splits an edge into two halves.

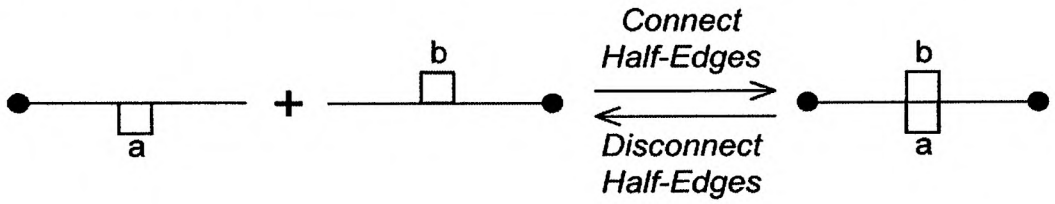

Figure 5.27 *Connect/Disconnect Half-Edges* operators.

```
function ConnectHalfEdges(e1, e2) {
e1.S:=e2;
e2.S:=e1;
}
```

```
function DisconnectHalfEdges(e) {
eSym:=e.Sym;
e.S:=e;
eSym.S:=eSym;
}
```

Table 5.24 *Connect/Disconnect Half Edges* operators.

Make / Kill Degenerate Edge

Make Degenerate Edge creates an edge that has the same bounding vertex at two ends (such an edge is called degenerate) (see Figure 5.28). A dual edge is also constructed and assigned to the primal edge. *P* is the bounding vertex in the primal, *D* – in the dual (see Table 5.25).

Kill Degenerate Edge is the reverse operator that destroys two half-edges of a degenerate edge.

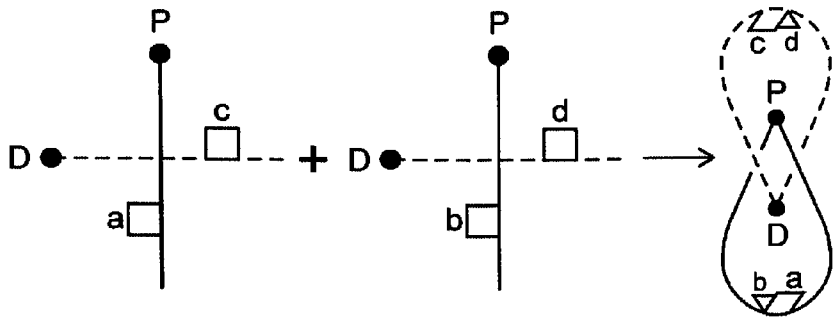

Figure 5.28 The *Make Degenerate Edge* operator.

```
function MakeDegenerateEdge(P, D) {
e:=MakeHalfEdge(P, D);
eSym:=MakeHalfEdge(P, D);
ConnectHalfEdges(e, eSym);
HalfSplice(e, eSym);
ConnectHalfEdges(e.Dual, eSym.Dual);
HalfSplice(e.Dual, eSym.Dual);
Result:=e;
}
```

```
function KillDegenerateEdge(e) {
KillHalfEdge(e.Sym);
KillHalfEdge(e);
}
```

Table 5.25 *Make/Kill Degenerate Edge* operators.

Make / Kill Edge

Make Edge creates two half-edges and connects them into an edge (see Figure 5.29) (*P1* and *P2* are edge bounding vertices; *D* is a dual vertex associated with a degenerated dual edge (see Table 5.26)). There is only one dual vertex, however the two ends of the dual edge are not connected in a disc cycle.

Kill Edge destroys the two halves of an edge.

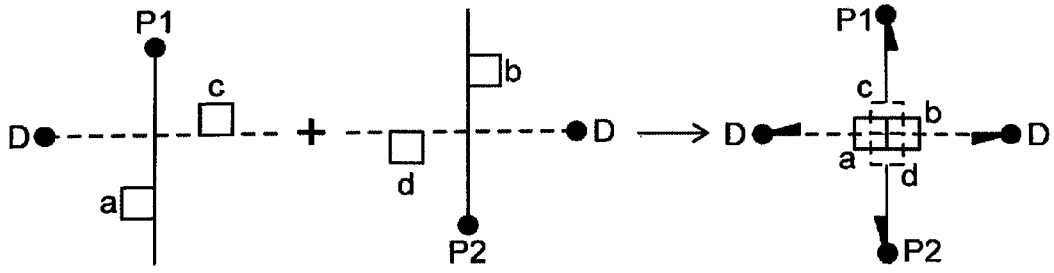

Figure 5.29 The *Make Edge* operator.

```
function MakeEdge(P1, P2, D) {
e:=MakeHalfEdge(P1, D);
eSym:=MakeHalfEdge(P2, D);
ConnectHalfEdges(e, eSym);
e.NF:=eSym;
eSym.NF:=e;
ConnectHalfEdges(e.Dual, eSym.Dual);
e.Dual.NF:=eSym.Dual;
eSym.Dual.NF:=e.Dual;
Result:=e;
}
```

```
function KillEdge(e) {
KillHalfEdge(e.Sym);
KillHalfEdge(e);
}
```

Table 5.26 *Make/Kill Edge* operators.

Make / Kill Half-Edge

Make Half-Edge reserves the computer memory for a DHE entity. This is the smallest element that can be created, however navigation is valid only with a full edge – single half-edges do not exist in a model. There are two parts of the dual half-edge: the primal and the dual (see Table 5.27 and Figure 5.30).

Kill Half-Edge release the memory occupied by a half-edge.

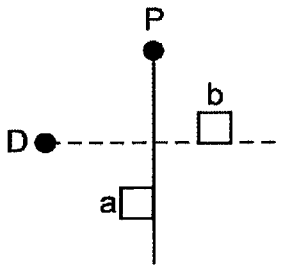

Figure 5.30 The dual half-edge element: *a* – a primal part, *b* – a dual part associated with the primal by the *D* pointer.

```
function MakeHalfEdge(P, D) {
eP:=TDHE.Create;
eP.NV:=eP;
eP.NF:=eP;
eP.S:=eP;
eP.V:=P;

eD:=TDHE.Create;
eD.NV:=eD;
eD.NF:=eD;
eD.S:=eD;
eD.V:=D;

eP.D:=eD;
eD.D:=eP;

Result:=eP;
}
```

```
function KillHalfEdge(e) {
e.D.Free;
e.Free;
}
```

Table 5.27 *Make/Kill Half-Edge* operators.

Relations between operators are shown in Figure 5.31.

Higher level operators are only permitted to call operators from lower levels. Some operators in level 3 use operators from level 1 – a two-level difference (i.e. *MVE*, *Join/Merge by Face*, and *Join/Merge by Vertex*). *MZEV* calls the *MEF* operator from the same level (see Table 5.9): some operations performed by *MZEV* are the same as in *MEF* – this operator is used as a shortcut.

The highest level (level 3) contains the *Euler operators* and *extended Euler operators*. Each pair of operators (base and reverse operators) is represented by a base operator (no reverse operators are shown in the diagram). (Level 4 would consist of the application program calling level 3.)

Compound operators from level 2 allow for construction and connection of edges (i.e. *Make Complex Edge* and *Complex Splice*) and faces (i.e. *Make Face* and *Sew*). This level may be used in applications for model construction – however their functionality would be limited (an alternative construction method using *Make Face* and *Sew* is described in Section 5.6).

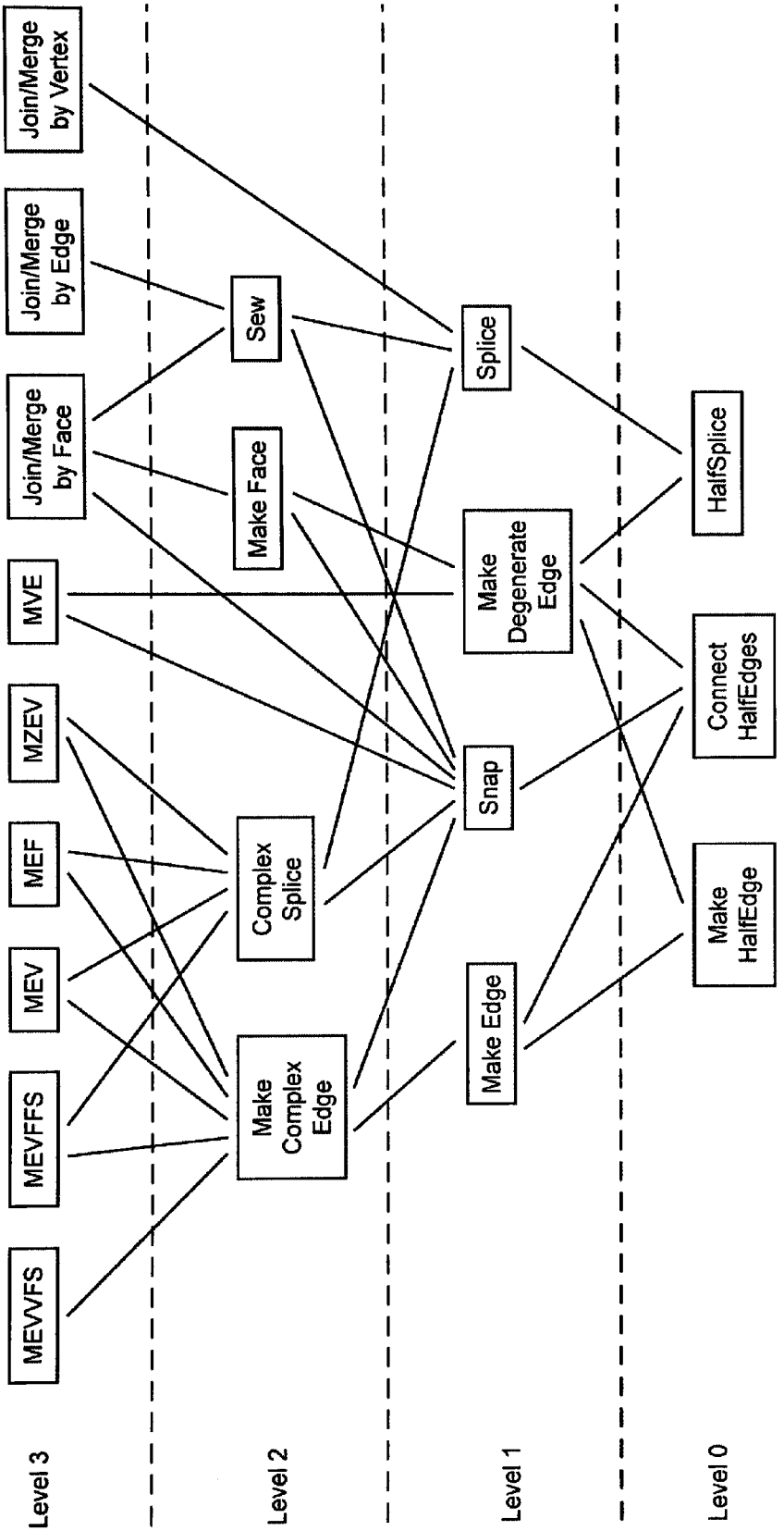

Figure 5.31 Construction operators organized in layers: level 3 – the Euler and extended Euler operators; level 2 – compound operators; level 1 – simple operators; level 0 – dual half-edge operators.

Operators in level 1 and 0 may directly change the DHE structure: *simple operators* in level 1 work with edges and call level 0 operators which work with dual half-edges: they change or assign the pointers and does not call other operators from the set. *Make Half Edge* is the only operator that reserves computer memory for the DHE.

5.5.6 Construction examples

A detailed example of a simple cell complex construction is presented below: this shows how to use the operators. First two cubes are constructed and then they are joined to obtain a cell complex. To make clearer pictures the dual is not presented in each step of the construction process.

There are 12 vertices which are used for a construction of two cubes:

$P_1(0, 0, 0)$, $P_2(0, 0, 1)$, $P_3(0, 1, 1)$, $P_4(0, 1, 0)$, $P_5(1, 0, 0)$, $P_6(1, 0, 1)$,
$P_7(1, 1, 1)$, $P_8(1, 1, 0)$, $P_9(2, 0, 0)$, $P_{10}(2, 0, 0)$, $P_{11}(2, 1, 1)$, $P_{12}(2, 1, 0)$.

Three dual vertices are also defined – one at infinity for external cells and two for the internal cubes:

$V_0(INF, INF, INF)$, $V_1(0.5, 0.5, 0.5)$, $V_2(1.5, 0.5, 0.5)$.

The vertices for the internal cells can be also calculated automatically as the centre of a cell – it has to be done at every step of the construction. However, the coordinates of the vertices are not important from the topological point of view – they are necessary, for example, for model visualization.

One of the possible Euler operators' sequences for a cube construction is as follows (see Figure 5.32 – dual space and the external cell are not shown to make the picture clearer):

e_1:=MEVVFS(P_1, P_2, V_1, V_0); e_2:=MEV(P_3, e_1.Sym); e_3:=MEV(P_4, e_2.Sym);
e_4:=MEF(e_3.Sym, e_1); e_5:=MEV(P_5, e_1.Next$_V$); e_6:=MEV(P_6, e_2.Next$_V$);
e_7:=MEV(P_7, e_3.Next$_V$); e_8:=MEV(P_8, e_4.Next$_V$); e_9:=MEF(e_6.Sym, e_5.Sym);
e_{10}:=MEF(e_7.Sym, e_9); e_{11}:=MEF(e_8.Sym, e_{10}); e_{12}:=MEF(e_5.Sym, e_{11}).

The result is a complex of two cells, internal and external, connected by the dual (see Figure 5.33). The internal and external cells are represented by the dual vertices V_1 and V_0 respectively – there is only one vertex V_0 in the model but it was 'multiplied' to represent dual edges as straight line segments going through faces of the cube.

The second cube is build using the same sequence but with different parameters:

e_{13}:=MEVVFS(P_5, P_6, V_2, V_0); e_{14}:=MEV(P_7, e_{13}.Sym); e_{15}:=MEV(P_8, e_{14}.Sym);
e_{16}:=MEF(e_{15}.Sym, e_{13}); e_{17}:=MEV(P_9, e_{13}.Next$_V$); e_{18}:=MEV(P_{10}, e_{14}.Next$_V$);
e_{19}:=MEV(P_{11}, e_{15}.Next$_V$); e_{20}:=MEV(P_{12}, e_{16}.Next$_V$); e_{21}:=MEF(e_{18}.Sym, e_{17}.Sym);
e_{22}:=MEF(e_{19}.Sym, e_{21}); e_{23}:=MEF(e_{20}.Sym, e_{22}); e_{24}:=MEF(e_{17}.Sym, e_{23}).

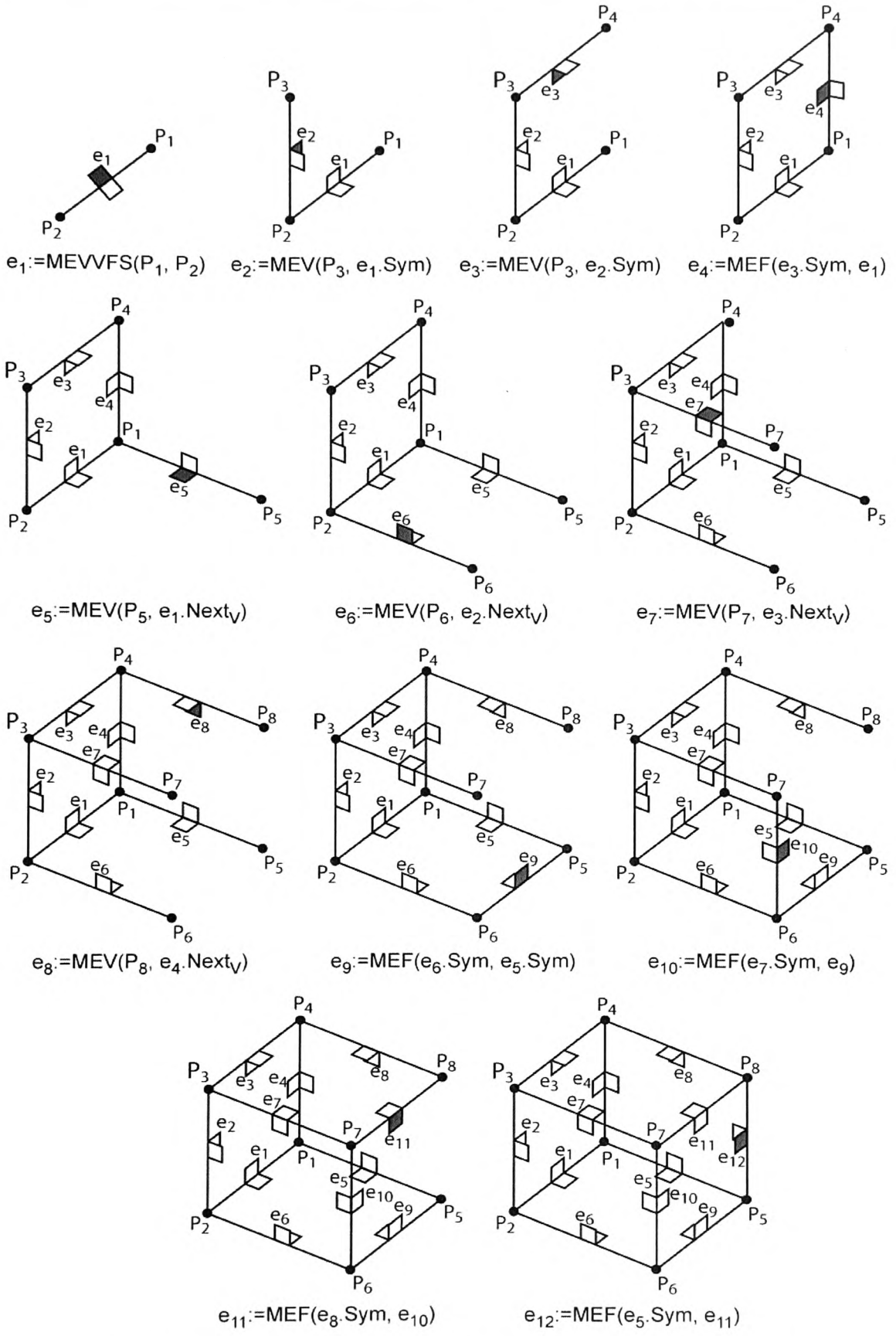

Figure 5.32 One possible Euler operators' sequence for a cube construction.

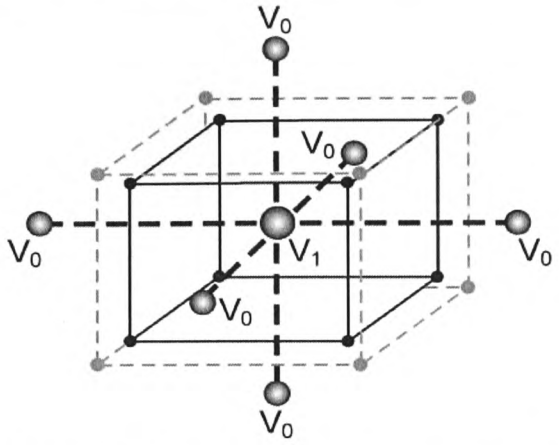

Figure 5.33 A cell complex consisted of internal (black solid lines) and external (grey dashed lines) cube connected by the dual (black dashed lines). V_1 represents the internal cell; V_0 represents the external cell.

After the cubes are built the *Join* operation can be performed. Two corresponding edges need to be connected – these edges will be in the adjacent relationship after the joining. The candidates are: e_9 from the first cell and e_{13} from the second cell (see Figure 5.34a). There is no connection between the cubes (however there are dual connections between internal and external cells not shown in the picture). *Join by Face* joins the two cells into one complex – their external cells (grey dashed lines) are combined into one cell:

$$\text{JoinByFace}(e_9, e_{13}).$$

The final model consists of three cells: two internal and one external. The internal cells represented by the dual vertices V_1 and V_2 are linked by a dual edge (bold dashed line) (see Figure 5.34b).

The above example is a simple cell complex construction method. The same result (two connected cubes) can be obtained in a different way – a box can be split into two connected cells (see Figure 5.35). This method and the set of Euler operators used is based on (Lee, 1999). The same set of vertices is assumed as before, plus a new dual vertex representing the original big box before the split:

$$V_3(1, 0.5, 0.5).$$

The Euler operators' sequence to build the origin box is also the same but the parameters are different:

e_1:=MEVVFS(P_1, P_2, V_3, V_0); e_2:=MEV(P_3, e_1.Sym); e_3:=MEV(P_4, e_2.Sym);
e_4:=MEF(e_3.Sym, e_1); e_5:=MEV(P_9, e_1.Next$_V$); e_6:=MEV(P_{10}, e_2.Next$_V$);
e_7:=MEV(P_{11}, e_3.NextV); e_8:=MEV(P_{12}, e_4.Next$_V$); e_9:=MEF(e_6.Sym, e_5.Sym);
e_{10}:=MEF(e_7.Sym, e_9); e_{11}:=MEF(e_8.Sym, e_{10}); e_{12}:=MEF(e_5.Sym, e_{11}).

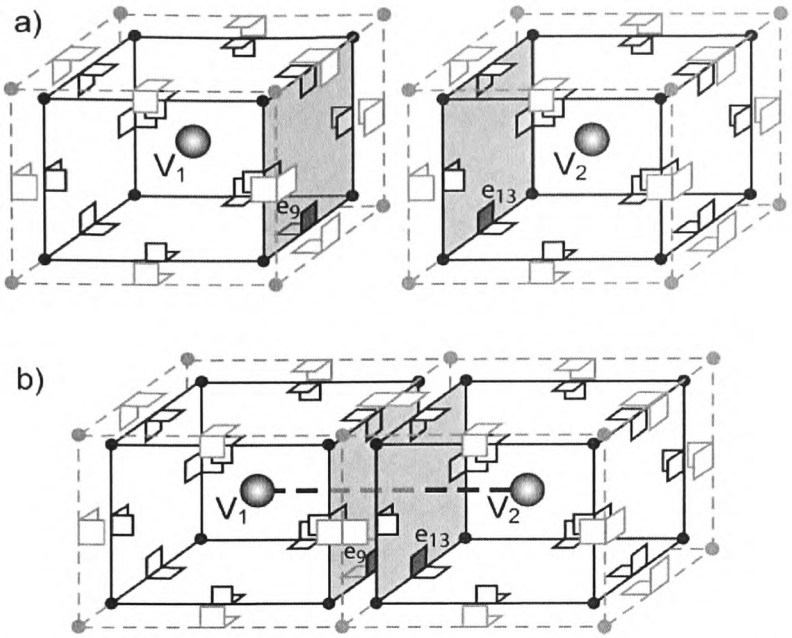

Figure 5.34 Example of joining cubes: a) two separate cells represented by dual vertices V_1 and V_2 are to be joined by adjacent faces represented by half-edges e_9 and e_{13}; b) cells after joining form a cell complex consisted of three cells.

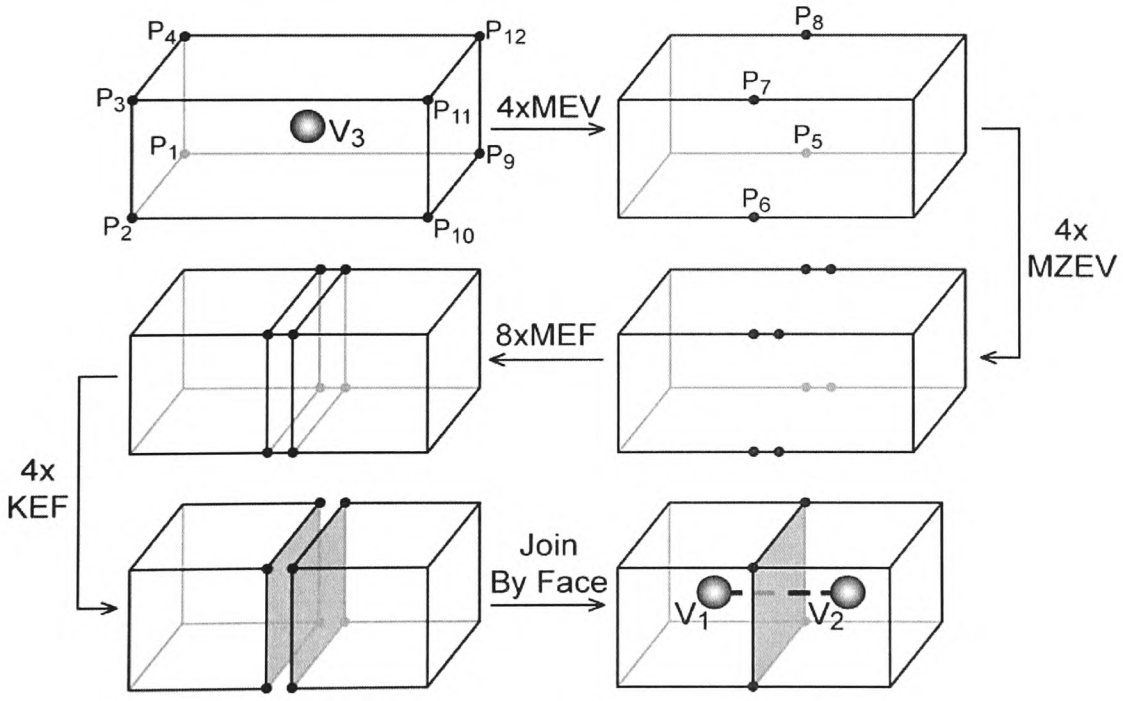

Figure 5.35 Splitting a box into a complex of two cells.

Then new vertices are added at the edge intersections with the cutting plane – P_4, P_5, P_6 and P_7:

$$e_{13}:=MVE(P_5, e_5);\ e_{14}:=MVE(P_6, e_6);\ e_{15}:=MVE(P_7, e_7);\ e_{16}:=MVE(P_8, e_8),$$

and duplicate these vertices using *MZEV*:

$$e_{17}:=MZEV(e_{13}, e_{13}.Next_V);\ e_{18}:=MZEV(e_{14}, e_{14}.Next_V);\ e_{19}:=MZEV(e_{15}, e_{15}.Next_V);$$
$$e_{20}:=MZEV(e_{16}, e_{16}.Next_V).$$

New edges that split the faces along the intersection are added with the cutting plane and repeat this operation for the duplicated vertices:

$$e_{21}:=MEF(e_6.Sym, e_{17}.Sym);\ e_{22}:=MEF(e_7.Sym, e_{18}.Sym);$$
$$e_{23}:=MEF(e_8.Sym, e_{19}.Sym);\ e_{24}:=MEF(e_5.Sym, e_{20}.Sym);$$
$$e_{25}:=MEF(e_{16}, e_{17});\ e_{26}:=MEF(e_{13}, e_{18});\ e_{27}:=MEF(e_{14}, e_{19});\ e_{28}:=MEF(e_{15}, e_{20}).$$

Finally the edges added before using *MZEV* are removed but this time the *KEF* operator is used, and then *Join by Face* is performed:

$$KEF(e_{17});\ KEF(e_{18});\ KEF(e_{19});\ KEF(e_{20});$$
$$JoinByFace(e_{21}, e_{26}).$$

After the box is split the new dual vertices (V_1 and V_2) should be assigned to all dual half-edges linked with half-edges of the first and second cell respectively. This step can be skipped if coordinates for dual nodes are calculated automatically.

The above example can be simplified using the extended operator *Splice by Face* (see Figure 5.36). After the original box is created (edges $e_1 - e_{12}$) and new intersection vertices added (edges $e_{13} - e_{16}$) all *MZEV*s are skipped and faces along the cutting plane are split: $e_{17}:=MEF(e_6.Sym, e_{13});\ e_{18}:=MEF(e_7.Sym, e_{14});\ e_{19}:=MEF(e_8.Sym, e_{15});\ e_{20}:=MEF(e_5.Sym, e_{16})$.

Then the list of edges which is used in *Split by Face* is created:

$$eList.Add(e_{17});\ eList.Add(e_{18});\ eList.Add(e_{19});\ eList.Add(e_{20});$$
$$SplitByFace(eList).$$

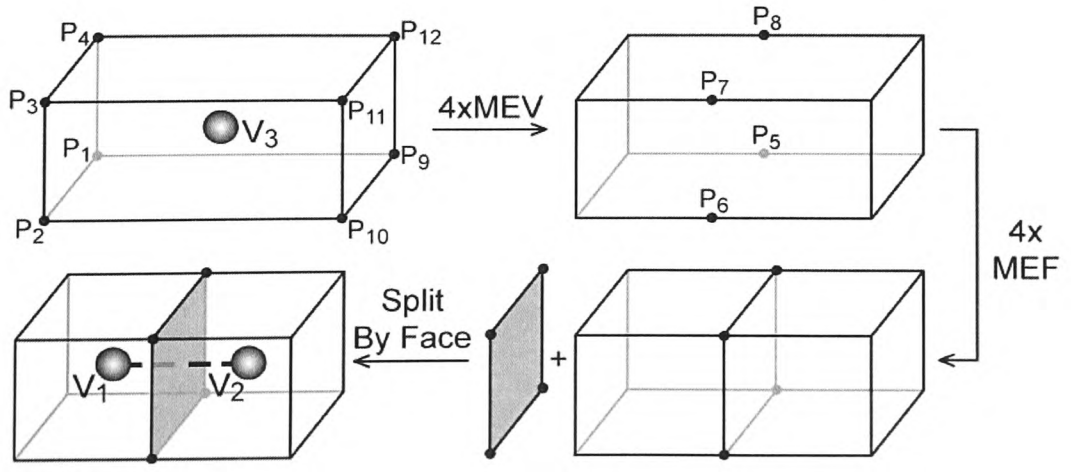

Figure 5.36 Splitting a box into a complex of two cells using the *Split by Face* operator.

Now dual vertices can be assigned to the cells as in the previous example (if they are not calculated automatically).

The result is the same as in the previous example but the number of steps and created edges is smaller.

5.5.7 Attributes

Semantic information specifies the status, functionality, meaning, usage or other characteristic of real objects (Zlatanova, 2000a). This may be stored in a model using attributes.

In the proposed models it is possible to assign information to all elements: cells (volumes), edges, faces, connections between cells, and vertices. However the only entities are edges and vertices, thus attributes can be assigned only to edges and vertices; the rest (i.e. volumes, faces, and connections) are represented by the dual of these two basic entities (i.e. edges and vertices).

For example, in the model shown in Figure 5.37 there are three connected boxes representing three adjacent rooms. Rooms are represented by dual vertices, thus an attribute describing a primal cell (e.g. a room name) can be assigned to a dual vertex; two other attributes 'Dijkstra distance' and 'next escape connection' are used by the dual graph traversal algorithm for escape route seeking. The same idea is applied for walls and connections between rooms – a primal entity attribute can be assigned to its dual counterpart: information about wall colour or door existence can be assigned to a dual edge representing the primal wall. It is worth mentioning here that all walls are double-sided: thus different attributes can be assigned to each side of a wall, as they very often have different finishing. A property of a connection between rooms (e.g. 'connection weight') is a dual entity attribute, and is assigned directly to an edge connecting these rooms. Any attribute, like an ID number, can be assigned to primal and dual edges and vertices.

Because a connection between two adjacent cells is represented as a bundle of edges (see Section 5.4.3), an attribute can be assigned to one of the edges in the bundle and be considered as an attribute of this bundle, or a reference to the attribute can be assigned to all edges in the bundle. Bundles are directed, so different navigation weights may be assigned to each direction.

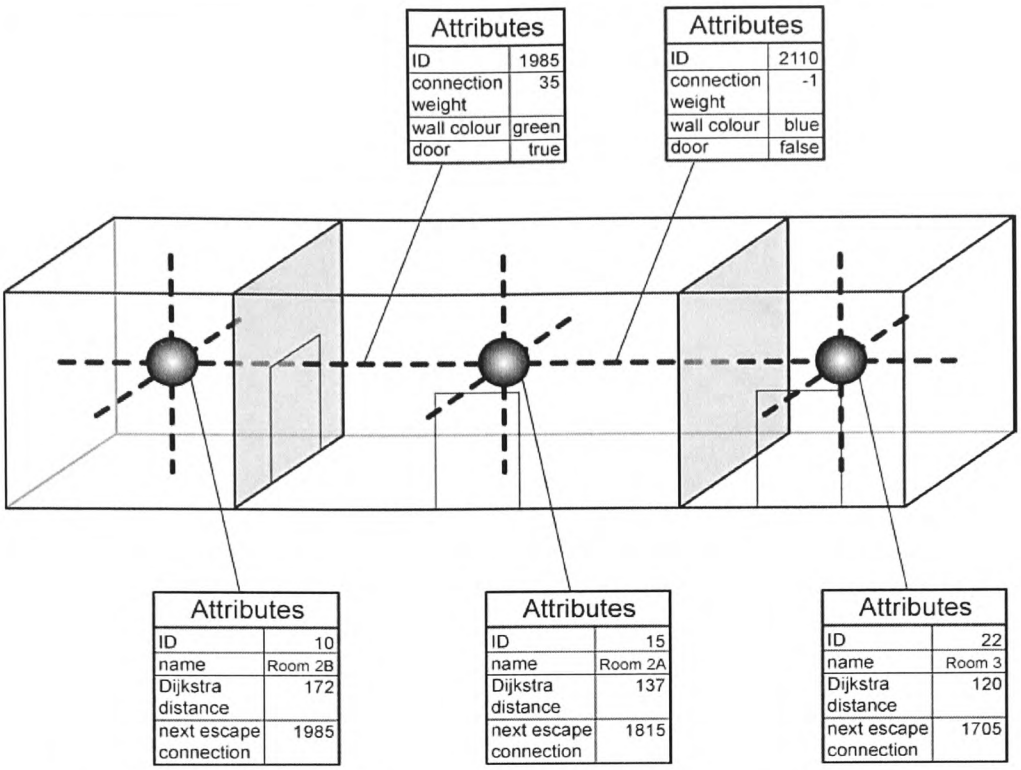

Figure 5.37 Attributes can be assigned to primal nodes and edges as well as to dual elements.

5.6 Cardboard & Tape (C&T) – a modified version

The construction method based on Euler operators presented in the previous section conforms to CAD systems. The author believes this can be adopted and used in systems to construct 3D models represented as cell complexes. However during the project a different method has been developed which is probably more intuitive. This is based on the construction of separate double-sided faces and 'sewing' them into polyhedra. It is called the 'Cardboard & Tape' method because the process is similar to the gluing of cardboard pieces using tape. This idea was presented by Boguslawski and Gold (2009a). The final result – a complex consisting of closed cells is the same but the intermediate steps are different.

These intermediate objects seem to be interesting: for example an open box, a fan of faces sharing an edge, the Möbius strip, etc. (See Figure 5.38.) It is possible to obtain these using the proposed Euler operators but this is not straightforward. For example, to create an open box, first a closed box is constructed, then one of the faces is merged with the adjacent face: *MergeByFace(e.Adjacent, e)*. There are 24 edges created (12 edges for each: internal and external cell) and in the merge operation four edges are removed (the four edges of the merged face). While using C&T only 20 edges are created and no edges are removed from a model.

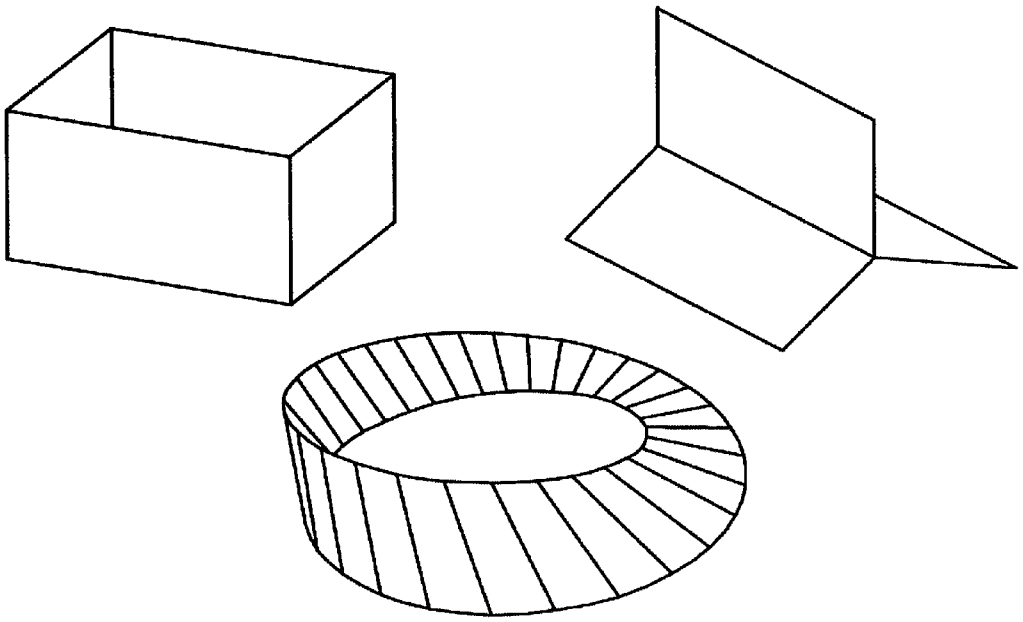

Figure 5.38 Examples of objects created with the Cardboard & Tape method.

Another example is a fan of faces: if Euler operators are used and for some reason only an internal cell without the external one is analysed it is necessary to remove this external cell from the model. After the fan of faces is built *MergeByFace(e, e.Sym)* is performed for each face from the external cell – in this way faces are removed from the model. C&T constructs only internal faces and sews them into a fan – no faces are removed. In the presented examples this is easier using the C&T method. This also requires a smaller number of new and deleted entities in a model (i.e. edges, cells, vertices).

The C&T method is based on the lower layer operators described in the previous section: faces of a model are created using *Make Face* and then they are connected with *Sew* to obtain cells. For example, to create a cube, eight square faces are built and the *Sew* operator performed 12 times – the result is a complex of two cells: one internal and one external cube connected by dual edges (see Figure 5.39).

The main limitation compared with the method based on Euler operators is the fact that single edges are not allowed – the simplest construction element is a face. This is a cost to pay for the ease of use (only two operators). However for some applications this is not a drawback. Examples are systems for modelling and manufacturing sheet metal products. Some of them use traditional solid modelling but recently a new approach has been developed: sheet metal parts are composed of flat faces (with zero thickness) joined by bend or weld edges (Lipson and Shpitalni, 1998). It seems that the C&T models fit perfectly to this approach.

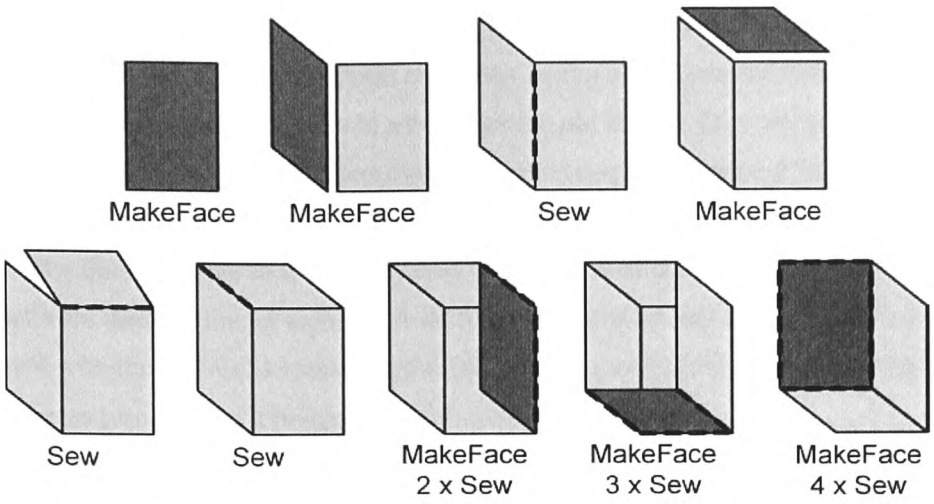

Figure 5.39 A cube construction process using C&T.

5.7 Simplified versions

The DHE data structure is powerful – it allows for non-manifold modelling (i.e. cells joined by a shared edge or vertex). However in many cases this functionality is not required – connection by a face is the only permitted connection between cells. For example, in building interior models adjacent cells are joined by common faces, and joining by an edge or a vertex is not allowed because only passages between rooms that can be used by people are taken into consideration. Another example of a 3D model where connections by only an edge or vertex are not allowed is tetrahedralization – space is tessellated therefore all cells must be joined by faces. A simplified version of the full DHE data structure that allows for storage space saving was developed during this research. Models with all cells connected only by faces can use this simplified version: cells may be connected by a common edge but connection by a vertex is not allowed. The main difference by comparison with the full version is that the N_F pointer is simulated by a combination of N_V and S – thus N_F can be removed. This reduces the number of pointers from ten to eight – the N_F pointer is removed from the primal and dual half-edges. Because the $Next_F$ navigational operator is based on the removed pointer (see Equation 5.3) it is necessary to modify this operator. A new version of $Next_F$ is defined in Equation 5.10. Construction operators which change the N_F pointer should be modified too: *Half Splice*, *Snap*, *Make Edge*, and *Make Half-Edge* – the code changing N_F should be removed.

5.10 $$e.Next_F = e.Sym.Dual.Next_V.Dual$$

Analysis of the storage space consumption for a simple cell complex is presented later in Section 5.9.

A further simplification saving storage space is also possible. The dual structure can be removed entirely from models where it is not used for advanced analysis (e.g. model

visualization). Volume elements (dual nodes) and connections between cells (dual edges) are no longer present in the model, thus attributes cannot be assigned to these entities. This simplification does not mean that cells in a complex are not linked. They are still connected and form a complex, however dual edges representing connections are removed. To keep the connections between cells the meaning of the *D* pointer is changed. It was used to link the primal with the dual structure in the full version. Since the dual is not present in the model adjacent cells are directly linked using *D*. A half-edge pointed by *e.D* in the simplified version was obtained with the *e.D.S.D* sequence in the full version (see Figure 5.40). It can be imagined that the dual was torn out and a bridge was built to link cells instead.

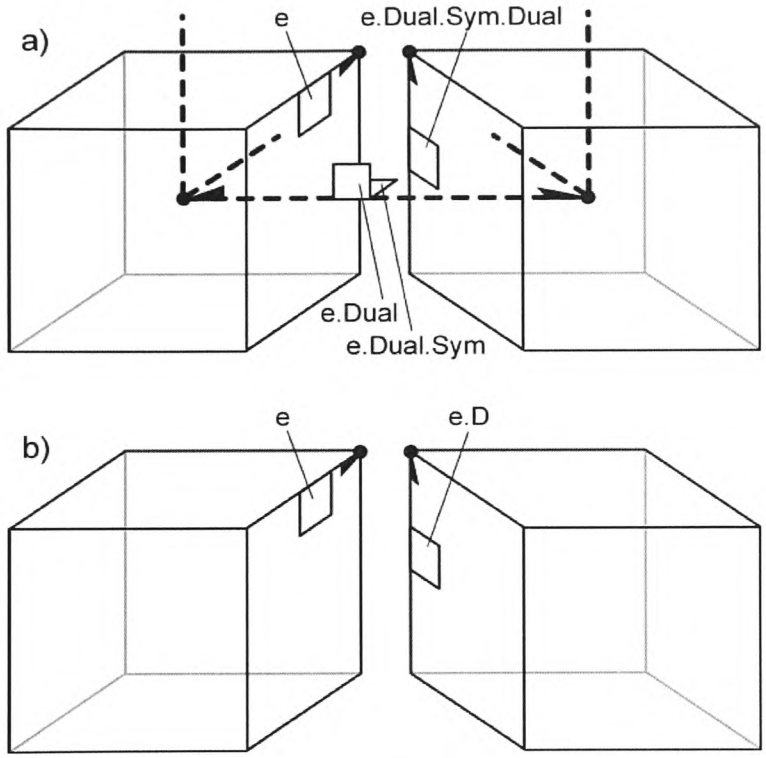

Figure 5.40 Connection between adjacent cells: a) the full version; b) the simplified version with no dual.

Because the *D* pointer points at the same primal structure, the navigational operators require some changes (see Equations 5.11 – 5.18). The *Dual* operator is removed from the new set.

5.11 $$e.Sym = e.S$$

5.12 $$e.Next_V = e.N_V$$

5.13 $$e.Prev_F = e.Next_V.Sym$$

5.14 $$e.Adjacent = e.D.Prev_F$$

5.15 $$e.Next_F = e.Adjacent.D$$

5.16 $$e.Prev_V = e.Sym.Next_F$$

5.17 $$e.Next_E = e.D.Next_V\ (e.Adjacent.Sym)$$

5.18 $$e.Prev_E = e.Sym.D.Prev_F\ (e.Sym.Adjacent)$$

Some of the construction operators from the lowest layer require changes too: *Complex Splice*, *Splice*, *Snap*, *Make Degenerate Edge*, *Make Edge*, and *Make Half-Edge*. However, modified versions of these operators are not presented in this work – the full DHE data structure is the most important and the simplified versions are presented to emphasize the data structure flexibility and to compare them with other data structures (see Section 5.9).

Another simplification, saving storage space by reducing the number of pointers necessary to represent each dual half-edge without losing any functionality, is that of merging a primal and dual half-edge into one element (they are permanently connected anyway). The DHE element is stored in one object in memory (primal and dual half-edges are combined into one object) instead of two objects (primal and dual half-edges are stored in two permanently connected objects). This solution reduces the number of references between objects thus the complexity of queries is reduced. This simplification reduces the number of pointers from ten to eight, because the *D* pointer used to link a primal with a dual half-edge is no longer necessary. The new TDHE class is shown in Table 5.28. The original DHE data structure described in detail in Section 5.5 is more memory consuming but directly illustrates the idea of duality which was one of the concerns in the project.

```
class TDHE {
V, DV: TVertex;
S, NV, NF, DS, DNV, DNF: TDHE;
}
```

Table 5.28 Simplified version of the *TDHE* class – pointers representing the primal and dual half-edge are stored in one class.

5.8 Limitations

The DHE can be used in various applications for 3D model representation. However this is a data structure and there are some limitations that may be important in some applications. Some of them will be described in this section.

There is no mechanism for an automatic level of detail change – if a very detailed element (for example a window with a handle, hinges and detailed frame) is present in a model it is not possible to automatically produce a simplified version (a polyhedron consisted of a few faces). CAD projects or specification sheets are sometimes very detailed but for model analysis or visualization these details are irrelevant – only a simple shape and semantic information is

important. The only easy solution to reduce the number of unnecessary details, that works only for objects consisted of several cells (for example if each part of a window is represented as a separate cell), is to merge all internal faces that are not adjacent to the external cell.

An interesting idea of a spatial indexing that could be used to change the level of detail was presented by Gold and Angel (2006). They consider Voronoi hierarchies but some elements of their work might be adopted, especially as they use the quad-edge data structure for model storage. This could be also combined with another idea of solid aggregation (Gröger and Plümer, 2010b): a hierarchical structure (a tree) is used to maintain the aggregation - atomic solids (e.g. whole rooms, parts of a room) are stored in leaves, while the root represents a complex object (e.g. a whole building). In the window example from the previous paragraph all small window parts would be stored in tree leaves while the whole window would be represented by a parent node. Depending on the level of detail, the detailed window or just the general shape of an aggregated object are displayed. A similar concept was presented by Coors (2003) where a tree structure is combined with a hierarchical level of detail – a high level of detail is visualized only near the viewpoint; with more distant objects (thus their importance is lower) the visualization quality is lower.

Another limitation of the proposed models is a lack of validation. The highest layer of operators formally described (see Section 5.5) – Euler operators – may produce invalid models. Euler operators do not check or fix the topological consistency of a model (Mäntylä, 1988). It is possible to implement a higher layer of operators that check that consistency before operators from the lower layer are performed: the geometry of a model should be taken into consideration. However the topological validation of 3D models is not easy (Ledoux and Meijers, 2010; Ledoux et al., 2009).

Simplified versions of the DHE data structure described in Section 5.7 save storage space but they are limited to models that do not include cells joined by vertices. In models which allow only cells joined by faces (e.g. decomposition of space, tetrahedralization) the simplifications may be appropriate and the storage efficiency may be improved.

The limitations described above are not inherent in the proposed models – the author believes that further research focused on applications will resolve these problems.

5.9 Comparisons

In this section the DHE and simplified versions are compared with other data structures. The author tried to find similar structures that are well described, and that could be used in similar applications. Three structures were selected: the coupling-entity (Yamaguchi and Kimura, 1995; Yamaguchi et al., 1991), the partial-entity (Lee and Lee, 2001), and 3D Navigable Data Model (Lee, 2007).

5.9.1 The coupling-entity – the feather

Probably the closest CAD data structure to the proposed approach is that of Yamaguchi et al. (1991) and Yamaguchi and Kimura (1995). A basic description was presented in Section 4.2. There are two groups of pointers in the structure: mate pointers (FM – fan mate, BM – blade mate, and WM – wedge mate) (see Figure 4.10a) and cyclic pointers (CCD – counter-clockwise disc, CCL – counter-clockwise loop, CCR – counter-clockwise radial, CD – clockwise disc, CL – clockwise loop, and CR – clockwise radial) (see Figure 4.10b – only counter-clockwise cycles are shown). The cycle pointers can be deduced from the mate pointers (Equations 5.19 – 5.24). Thus the full set of nine pointers can be reduced to three and mate pointers are used exclusively.

5.19 $$CCL(e) = FM(BM(e))$$

5.20 $$CCR(e) = WM(BM(e))$$

5.21 $$CCD(e) = WM(BM(FM(e)))$$

5.22 $$CL(e) = BM(FM(e))$$

5.23 $$CR(e) = BM(WM(e))$$

5.24 $$CD(e) = FM(BM(WM(e)))$$

Because of this reduction the functionality of their model is limited – no more than one disc cycle around a vertex can be represented with the feather data structure (Yamaguchi and Kimura, 1995). Thus some topological relations are not possible in their model, for example joining two objects at a vertex (see Figure 5.41). Here there are two disc cycles that need to be joined ($e_1 \rightarrow e_2 \rightarrow e_3$ and $e_4 \rightarrow e_5 \rightarrow e_6$) – one disc cycle from each cell. But this is not possible with the mate pointers only – all mate pointers are already used to connect the neighbouring entities of a single cell. The solution to that problem that allows for this non-manifold case can be the adding of the disc cycle pointers back to the feather (CCD and CD). Thus navigation around the shared vertex would be explicit and the loop and radial cycles could be still deduced using the mate pointers.

The DHE is a more complex data structure than the coupling-entity. Non-manifold cases like the one presented in Figure 5.41 can be modelled. It is also possible to simulate the feather: all the pointers can be derived from the DHE. The wedge mate pointer *WM(e)* is explicitly represented with *Sym* (*S* pointer) (Equation 5.25); fan *FM(e)* and blade *BM(e)* mates can be represented as a sequence of basic DHE pointers or navigation operators (Equations 5.26 and 5.27):

5.25 $$FM(e) = e.Dual.Sym.Dual: e.D.S.D$$

5.26 $$BM(e) = e.Adjacent: e.D.N_F.D.S$$

5.27 $$WM(e) = e.Sym: e.S$$

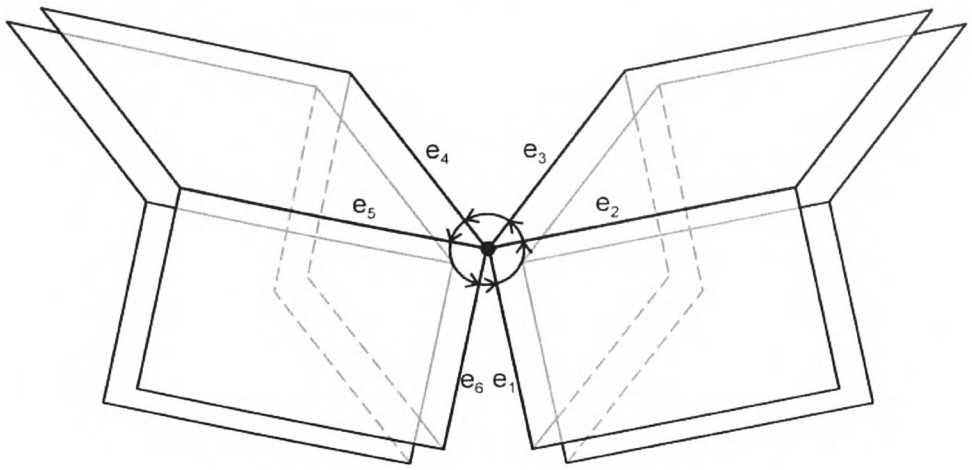

Figure 5.41. Joining two cells at a shared vertex.

The cycle pointers can be derived (Equations 5.28 – 5.33) from mate pointers, but there are also DHE pointers that can be used explicitly:

5.28 $$CCL(e) = e.Next_F : e.N_F$$

5.29 $$CCR(e) = e.Next_E : e.D.N_F.D$$

5.30 $$CCD(e) = e.Next_V : e.N_V$$

5.31 $$CL(e) = e.Prev_F : e.D.S.N_F.D.S$$

5.32 $$CR(e) = e.Prev_E : e.S.D.N_F.D.S$$

5.33 $$CD(e) = e.Prev_V : e.D.N_V.D$$

It was shown above that a model using the feather as a basic element can be simulated with the DHE data structure. It is also possible to show that a simplified version of the DHE exists and this is an equivalent of the feather. Thus the feather is a subset of the DHE. In the simplified version there is no dual and the N_F pointer is removed from the structure – there are N_V, S and D pointers left (the V pointer is not taken into consideration, because it does not have any topological function). Because the dual is no longer available in the model, the D pointer has a different meaning and does not point to the dual half-edge – it points to a half-edge in the adjacent face. This connection is the same as the fan mate connection in the feather. Equations 5.34 – 5.36 show the relations between the two models. The mate pointers can be defined with DHE pointers:

5.34 $$FM(e) = e.D$$

5.35 $$BM(e) = e.Adjacent : e.D.N_V.S$$

5.36 $$WM(e) = e.Sym : e.S$$

The cycle pointers can be derived from the mate pointers as shown in Equations 5.19 – 5.24 replacing the mate pointers with DHE equivalents (Equations 5.34 – 5.36).

Because the simplified version is an equivalent of the feather a reverse translation is possible (Equations 5.37 – 5.39):

5.37 $$e.D = FM(e)$$

5.38 $$e.N_V = WM(BM(FM(e)))$$

5.39 $$e.S = WM(e)$$

Unfortunately the simplified DHE has similar limitations to the feather. Joining two cells at a shared vertex produce a model which is not valid. However all edges sharing the same vertex can be joined in one cycle using the N_V pointer: $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_6$ (see Figure 5.41), but navigation is not valid. For example the CL (clockwise around a face) cycle is defined in Equation 5.22. The fan and blade mates can be replaced with the DHE equivalents (Equations 5.37 and 5.38). Thus CL is defined as $N_V.S$. The inability to navigate around a face appears in two places: e_3 and e_6. The e_3 edge will be used in the example. $e_3.N_V.S$ points at the opposite end of e_4 but in the valid face loop the next element is the other end of e_1. This error is caused by defining the loop cycle using the disc cycle. Unfortunately this does not work for the case presented in Figure 5.41 – the disc and loop cycles should be independent. However, in a cell complex decomposing 3D space where all the cells are joined by shared faces other methods can be used to navigate between edges sharing the same vertex. A single cell in a complex is a 2-manifold, and only one disc cycle is necessary to navigate around a shared vertex. Disc cycles in other cells are not joined together, but navigation through shared faces is possible, and access to these cycles is possible.

It was shown in this section that the DHE data structure can be simplified to an equivalent of the feather coupling-entity. Not all non-manifold cases can be managed with this simplified version: two cells joined at a vertex are not allowed. An extra cycle pointer and the dual are necessary for valid navigation in a model. Another difference and a big advantage over the coupling-entity is that the DHE (the full version) is able to represent a cell/volume with a single dual vertex, and a face entity with a bundle (of edges). Thus for example attributes may be assigned to any node, edge, face, or volume entity in either the primal or dual space.

The full system (with the dual) developed during this research can represent certain 3D degeneracies in the model. Adding a 'bridging edge' in each face permits the construction of a hole through a shell and adding a 'bridging face' allows modelling of a completely enclosed cavity. (Adding these elements means a single connected graph is formed, and it may be navigated, queried and edited.)

5.9.2 The partial-entity structure

The partial-entity structure introduced by Lee and Lee (2001) is a compact data structure that comes from the radial-edge (see section 4.2). Data storage is reduced significantly (by about 50%) without the loss of time efficiency.

A model consists of one or more regions that are bounded by shells – there is always one infinite open region and zero or more closed regions. Regions are equivalent to cells in the DHE structure – a model consists of one external cell of infinite volume and zero or more internal cells. Two incident regions share a face – each side of the face is a part of each region. Faces are bounded by loops of edges – there is one peripheral loop and zero or more hole loops. Face-, edge-, and vertex-uses of the radial-edge were replaced with new entities respectively: p-face, p-edge, and p-vertex. The differences between these two representations allow for decreasing storage space without loss of functionality.

A comparison of the radial-edge with the partial-entity structures is presented by Lee and Lee (2001): analyses of time and storage complexity were taken into consideration. Time efficiency tests are not performed here, however the DHE storage requirements can be compared to the results obtained for the radial-edge and partial-entity structures. Because the DHE data structure is developed to maintain cell complexes (but not only that), in this comparison the case of a cell complex consisted of 1,000 cubes (10×10×10) is used.

The total size taken by the model constructed using the radial-edge is 1,457,303 bytes, and for the partial-entity – 644,192 bytes (Lee and Lee, 2001). The same restrictions are used in the calculation: the size of the field storing pointers is four bytes; fields for attributes or geometric data are not taken into consideration (only storage for topology is calculated); flags or lists are not used in the DHE structure. The analysis will be carried out for three cases: the full DHE version, and two simplified DHE versions.

There is one cell complex in the model: there are 1,000 cubes in the complex; each cube consists of 12 edges; each edge is represented by two dual half-edges; each DHE contains five pointers in each space – the primal and dual (that gives ten pointers for each DHE); each pointer takes four bytes. There is also one external cell present in the model – one big cube enclosing all internal cubes. Each face of the external cube is split into 10×10 grid of squares corresponding to the external faces of the internal complex – that gives 1,200 edges in the external cell.

Storage space for the internal complex equals: 1,000 cubes × 12 edges × 2 DHE × 10 pointers × 4 bytes = 960,000 bytes; and for the external cell: 1,200 edges × 2 DHE × 10 pointers × 4 bytes = 96,000; in total 1,056,000 bytes. This score locates the DHE structure between the radial-edge and partial-entity structures.

It should be noticed that in the full version the situation that two cells are joined by a vertex, and volumes (dual nodes) are stored explicitly can be managed. However if all cells in the

analysed model are joined by faces – the simplified version without the face loop pointer (see Section 5.7) may be more suitable. The number of pointers in the DHE is decreased from ten to eight. Thus the storage space required for the model is calculated as follows: 1,000 cubes × 12 edges × 2 DHE × 8 pointers × 4 bytes =768,000 bytes; and for the external cell: 1,200 edges × 2 DHE × 8 pointers × 4 bytes = 76,800; in total 844,800 bytes. This gives a 20% space saving.

Further simplification is even less space consuming if one does not need to use volume entities. The dual structure is removed (see Section 5.7) and all connections between cells in the model are stored in the primal. Thus only four pointers are sufficient to store topology information. The storage space required for this case is: 1,000 cubes × 12 edges × 2 DHE × 4 pointers × 4 bytes = 384,000 bytes; and for the external cell: 1,200 edges × 2 DHE × 4 pointers × 4 bytes = 38,400; in total 422,400 bytes. The DHE data structure requires about 30% storage size of the radial-edge and 65% of the partial-entity for the cell complex representation.

Table 5.29 shows detailed storage requirements of the presented comparisons.

The DHE structure is flexible. The full version with the dual graph included is useful when information needs to be stored for volumes and an explicit representation of connections between cells is important. A cell complex traversal using the dual graph is easier and the implementation of some algorithms (e.g. the Dijkstra algorithm) is straightforward.

To save storage space, the simplified version can be used for preliminary model construction, and then this can be expanded to the full version when more advanced analysis of the model is required.

Data structure	Total size	Ratio (x/RE)	Ratio (x/PE)
Radial-edge (RE)	1,457,303B	100%	226%
Partial-entity (PE)	644,192B	44%	100%
Full DHE	1,056,000B	72%	163%
Simplified DHE (no N_F pointer)	844,800B	57%	131%
Simplified DHE (no dual)	422,400B	28%	65%

Table 5.29 Storage requirements for a cell complex of 1,000 cubes (10×10×10).

5.9.3 3D Navigable Data Model (3D NDM)

The building interior models based on the DHE are closely related to the models obtained with the 3D NDM (Lee, 2007) described in Section 2.7. Both the 3D NDM and the DHE models are based on the Poincaré duality and graph theory; however there are some differences between them:

1. 3D NDM: A graph of connections between rooms is computed from the geometry; in a building model reconstruction process the horizontal and vertical adjacency relations are defined in two separated processes (Lee and Zlatanova, 2008).
   DHE: The direction of adjacency relations is not important thus complex buildings may be modelled – two adjacent rooms are joined by a shared face (if the faces do not fit one to another they are automatically split, thus for example two small rooms can share one face with a third big room). Also connections between rooms are reconstructed automatically along with changes of the geometry – for example if a wall between rooms is removed (destroyed) and two rooms are merged into one, the connection between the original rooms is removed and a new dual node representing the new bigger room is introduced: the rest of connections with other adjacent rooms are preserved, thus this modification is local and automatic.
2. 3D NDM: A solid (room) is transformed to a node in a dual graph; a common face between adjacent solids is transformed into an edge (Lee and Zlatanova, 2008). There is no dual representation of a node and edge, however the Poincaré duality assumes that a primal node is transformed to a cell, and an edge to a face, in dual space.
   DHE: The complete Poincaré duality is implemented: a primal solid is represented as a node in the dual graph: a face – as an edge; an edge – as a face; a node – as a solid. That makes the DHE representation more general: it can be used in other applications than building interior modelling.
3. 3D NDM: A logical network representing all the connections between rooms in a building is then transformed into a geometric network model which is useful for shortest path algorithms (for example a long corridor is not represented as one node but as a long line segment, then the nodes representing adjacent rooms are connected with this line segment along the shortest distance).
   DHE: The logical network is the final representation in dual space: however, long corridors may be split into smaller parts in such a way that the geometric network model will be identical with the logical network (the details of this method are not described in this work).

6 Applications*

It is demonstrated in this chapter how the DHE data structure can be useful for the modelling, management and analysis of 3D buildings.

With the newly adopted Open Geospatial Consortium (OGC) CityGML it is possible to represent the different aspects of three-dimensional (3D) city models (Kolbe, 2009; OGC, 2008). This representation can be multi-scale since five levels of details (LODs) for the same city can be stored: from LOD0 where for example the terrain and the transportation network are stored, to LOD4 where buildings have detailed roof structures, windows, rooms and even pieces of furniture. Extensions to the base model are possible for specific applications (Coors, 2008): there exist extensions for floods (Schulte and Coors, 2008) and energy management. However, while CityGML provides great flexibility for the storage and the representation of 3D buildings, the spatial data model used currently – the Geography Markup Language GML (OGC, 2007) – has limited capabilities for storing explicitly topology, and offers no tools/algorithms to build the topological data model. In most cases only the geometry of the 3D models is stored. While the connectivity information between the elements of the model can always be extracted on-the-fly, for many applications it is highly desirable to have them explicitly stored in a topological data structure (Ellul, 2007; Ellul and Haklay, 2006; van Oosterom et al., 2002; Zlatanova et al., 2004). Examples of such applications are when spatial analysis functions and/or dynamism is involved: updating the model in real-time, routing for emergency situations, etc.

* This chapter is largely based on Boguslawski, P., Gold, C.M. and Ledoux, H., 2011. Modelling and analysing 3D buildings with a primal/dual data structure. ISPRS Journal of Photogrammetry and Remote Sensing, 66(2): 188-197.

The DHE permits one to construct, represent and analyse the buildings (and their interiors) of 3D city models and to store explicitly: 1) the geometry of the elements of the model (the different buildings, the rooms, the terrain, etc.); 2) the topological relationships between the elements of different dimensionality (so we can navigate from a given point to all the connected faces, from polyhedron to polyhedron, etc.); 3) the attributes for any elements of the model (points, line segments, faces and polyhedra). This was achieved by simultaneously storing both the primal and dual subdivisions of a 3D city model (it was assumed that the model divides the 3D space into different polyhedra and that there is one 'universe' polyhedron). Duality, which is described in Section 3.2, is a concept that implies that two subdivisions are inter-connected: they represent the same thing but from a different point of view.

6.1 Model representation

A building in general consists of several connected rooms that have a volume (corridors, office, storage spaces, etc. are considered as rooms too), so they are represented by primal cells. The geometry of a room can be easily modelled with the edges and nodes of a cell; relations between adjacent rooms can be represented with dual edges connecting cells. Relations can be described in terms of the access level from one room to the adjacent one: access to the next room is easy by a door; otherwise the next room is not accessible because of a wall, but maybe the wall is thin and with special equipment a hole can be made; finally, perhaps it is not possible to get directly to the next room, because the wall is made of concrete. This is an example of a basic set of attributes that can be assigned to connections between rooms and then used as weights in graph traversal algorithms (e.g. the Dijkstra algorithm).

Rooms are not the only objects in a building that are important. Walls, doors, windows, installations etc. are essential in many applications and can also be included in a model. They can also be represented as cells with geometry and volume, and attributes can be assigned to them. Further analysis can answer questions about a building structure: are there any pipes or wires in the wall between rooms A and B; is the door one- or two-leafed, etc.

Two approaches can be distinguished:

Type 1: Rooms are not the only objects in a model: walls, doors, windows, installations and other objects are represented with 'thick' cells too – non-zero volumes can be calculated from the geometry of the objects.

Type 2: Only rooms have a non-zero volume; other objects can be present in a model, but they are 'flat' (see Figure 6.1). Adjacent rooms are connected directly – there is no wall in between; they can also be connected by doors that are represented as double-sided flat faces. The volume of a flat object is zero, but there is still a dual node for this object.

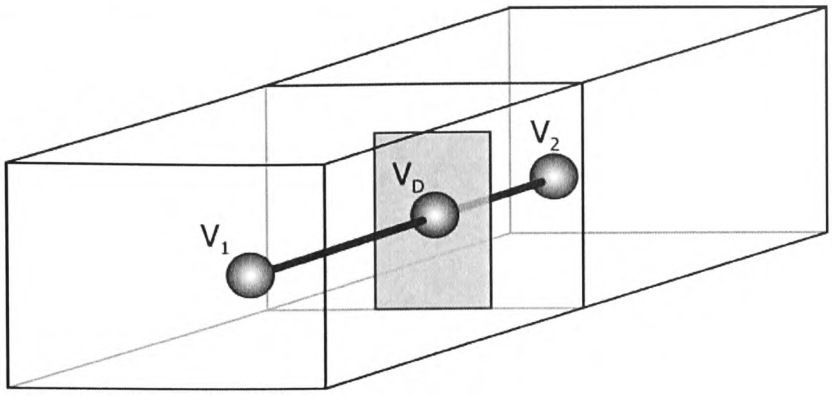

Figure 6.1 Two adjacent rooms V_1 and V_2 with the 'flat' door V_D in between. Only the connection through the door is taken into account in escape route planning; however a direct connection through a shared wall between V_1 and V_2 exists in the model (not shown in the picture).

6.2 Building model construction

To demonstrate the correctness of the presented methods a prototype was developed. The Delphi programming language was used to write it and an in-house graphics engine based on OpenGL for visualization. Two building models with two different model types: Type 1 and Type 2 were reconstructed.

3D building interior datasets are not easily available, but with the recent adoption of CityGML as an OGC standard, it should be possible to obtain more and more buildings in that format (which uses GML for the 3D geometry representation). As previously explained, the GML mechanism to represent topology has weaknesses and is seldom used; the DHE is used instead. It can be detected if cells are adjacent based on their geometry, and cells can be easily connected by adjacent faces, but the real world is not so simple. Some cells need to be edited before they can be connected. Not every wall in a building is shared by exactly two rooms (e.g. one long corridor may have many adjacent rooms). Adjacent rooms can have walls of different shapes or sizes. To solve this problem a boundary intersection module is required. During the construction process the locations of two adjacent cells are checked, new edges are added if necessary, and then connected (Figure 6.2). Topological relations between objects in 3D space were presented by Egenhofer (1995) – the relationship between two cells can be described with eight names: disjoint, meet, contains, covers, inside, coveredBy, equal, and overlap. Only the meet and disjoint relationships are important in this research. Overlapping of cells is not tested; datasets have to be validated first using different methods.

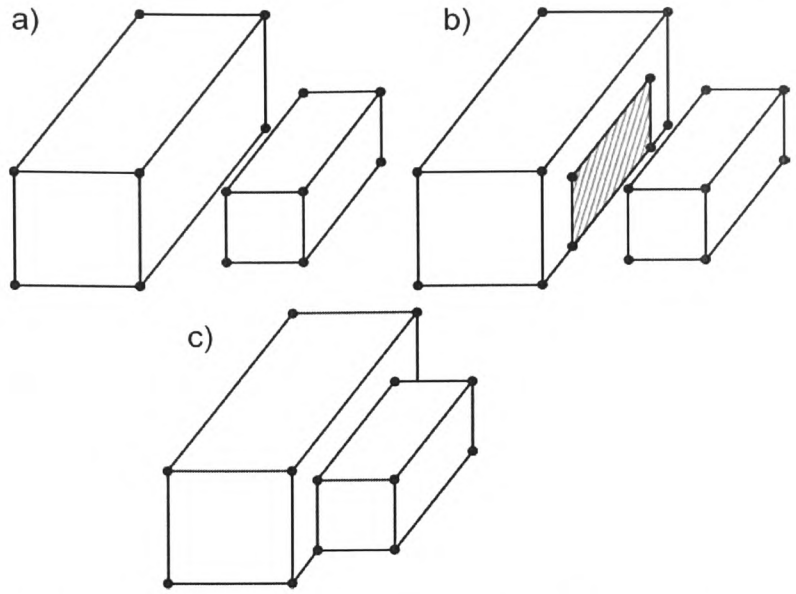

Figure 6.2 Boundary intersection module: a) two adjacent cells; b) new edges added to a bigger face create a new adjacent face; c) two cells connected.

The first reconstructed model was a simple house (see Figure 6.3). This is an example of *Type 1* – with walls, ceilings and windows represented as thick cells. The original dataset stored in CityGML format is available from the official CityGML webpage (www.citygml.org). Good quality data in this set is valid and no extra cleaning is necessary. The CityGML LOD4 is included in the file; this means that data describing the interior with rooms, walls and even furniture is present. Besides the geometry there is also semantic information included: there are sections in the file representing single objects with their function (e.g. this cell represents a room; this set of faces represents a wall, window, door, stairs, etc.).

In the current version of a CityGML format import application developed during this research, holes and cavities are not detected automatically (this is a subject for future work), thus windows (that should be recognized as holes in walls) are separate complexes not connected with the rest of the model.

The second model – two buildings from the University of Glamorgan campus (see Figure 6.4a) connected by an above-ground passage (see Figure 6.4b) – is an example of *Type 2* (no thick walls between rooms; doors are flat). In total there are over 1,300 cells. Cells may be rooms, doors or corridors. The model, and its use for escape route planning, was prepared within two weeks. This was reconstructed from scanned paper plans (see Figure 6.5a). These plans were used as a raster background in AutoCAD – subsequent floors were put into layers at different heights (the distance between layers was set at an arbitrary room height) (see Figure 6.5b). All rooms were manually traced (vectorised) and were represented as polygons (see Figure 6.5c). Doors, represented as line segments were put into a separate layer. Then all layers

were extruded (see Figure 6.5d): the room layers were adjusted to fit on top of each other (the extrusion height was the same as the distance between raster layers containing floor plans); door layers were extruded with a smaller arbitrary height. This produced a set of individual cells, one for each room – but not connected together (see Figure 6.5e). The model was imported to Autodesk 3DS Max, where labels (i.e. room number and name) were attached to each cell. The final model (still represented as a set of separate polyhedra) was exported to the OBJ format that was used in the final application.

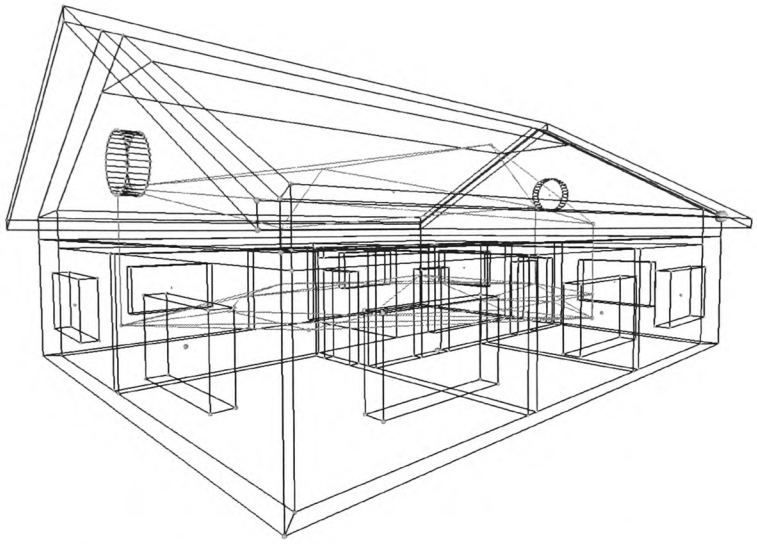

Figure 6.3 Model of a house reconstructed from the CityGML format using the DHE data structure. Walls, doors, and windows are represented as thick cells (with non-zero volume). Source: www.citygml.org

All the connections between adjacent rooms were set during the construction process using the DHE and Euler Operators, with both the primal and the dual graph being updated simultaneously – geometric intersection testing routines were used to check for adjacency. This model was created with the intention to use it in emergency systems for finding escape routes from a building, thus only adjacency by a face is taken into account (only navigation through faces is possible); however, cells can be connected by a shared edge or vertex. Because all adjacent cells are connected and we want to avoid navigation through solid walls, weights are assigned to connections between rooms; they describe how difficult is to move to the next room: an infinite value means no access, any other (positive) value is calculated from the geometric distance between the dual nodes representing adjacent cells. It is not easy in the plans to check a door's existence between rooms, and to input this information manually into the model. Doors were therefore modelled as well – but with zero thickness to put them in between two adjacent cells. Since doors are in the model weights can be calculated and assigned only if two rooms are connected by a door.

As the original objective was escape route planning, the local terrain close to the buildings (Boguslawski and Gold, 2009b) needs to be described – e.g. for assembly points. This was achieved by adding thin cells (perhaps concrete paving) to the model (light-grey cells in Figure 6.4a), allowing navigation outside the building.

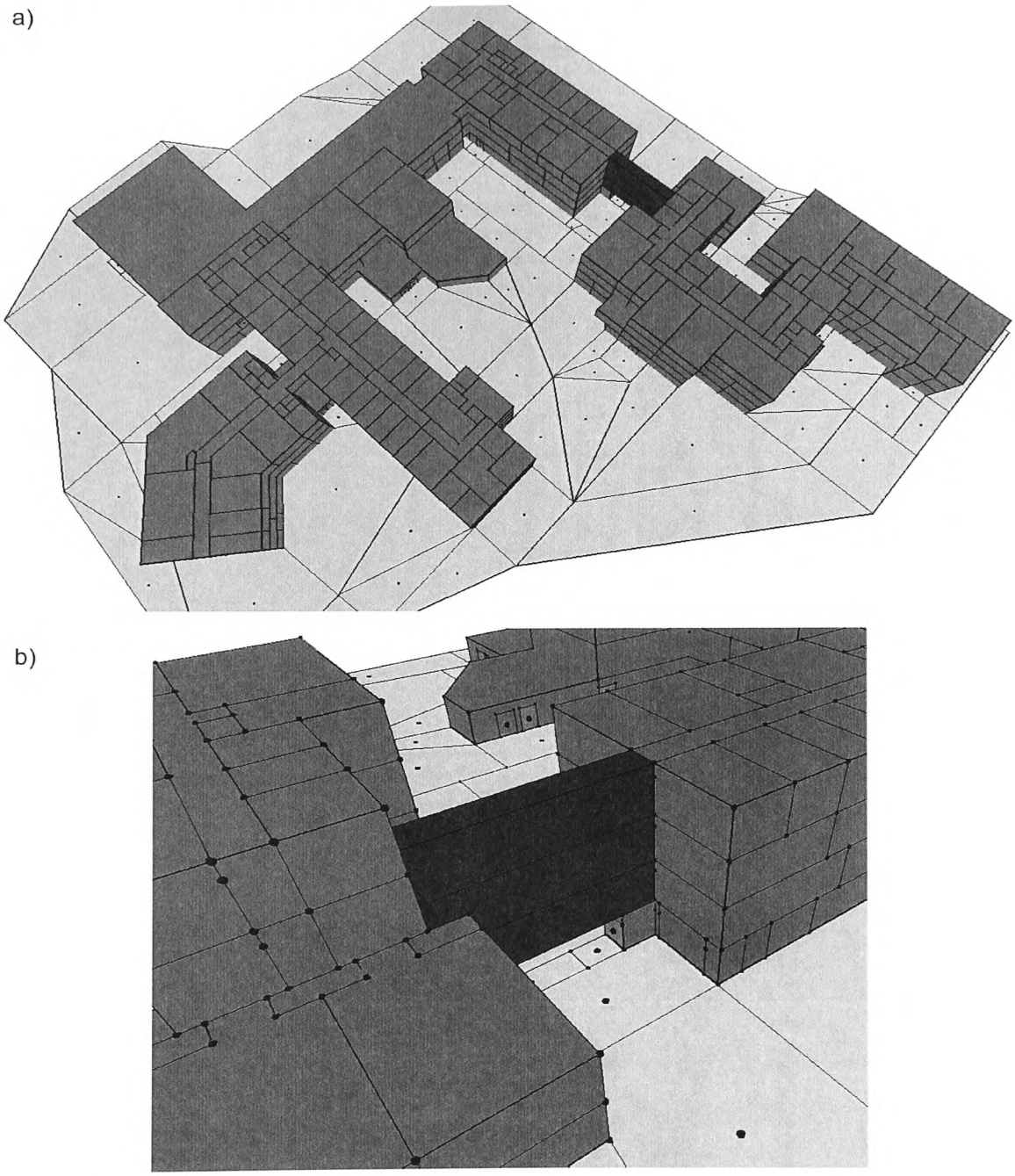

Figure 6.4 Two buildings from the University of Glamorgan campus connected by an above-ground passage modelled using the DHE data structure: a) light-grey cells represent terrain, grey cells represent rooms, dark-grey cells represent the above-ground passage between buildings; b) an above-ground passage between two buildings – dark cells.

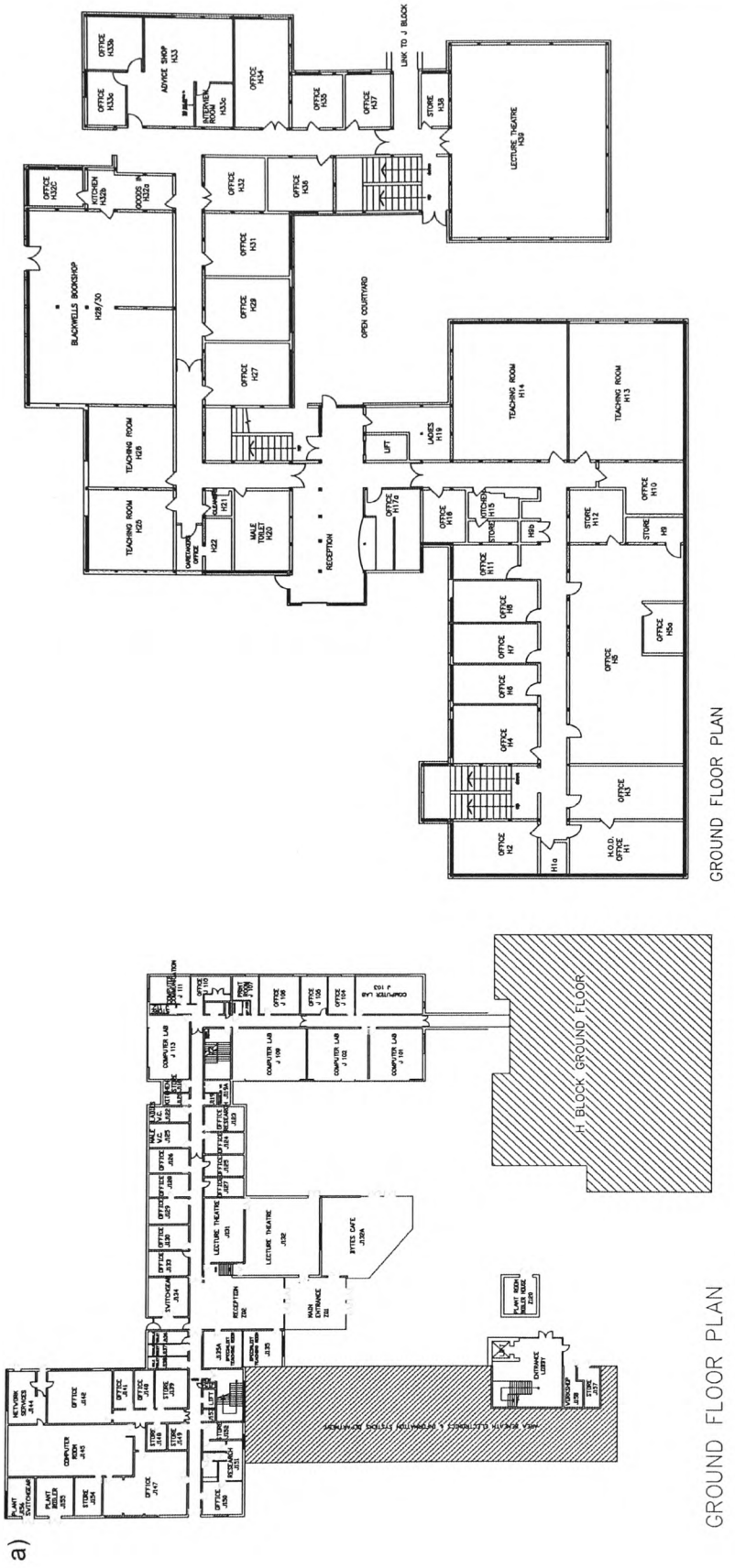

Figure 6.5 The model construction steps (H and J buildings from the University of Glamorgan campus): a) scanned paper plans.

Figure 6.5 The model construction steps (H and J buildings from the University of Glamorgan campus): b) raster backgrounds for all floors organized in layers.

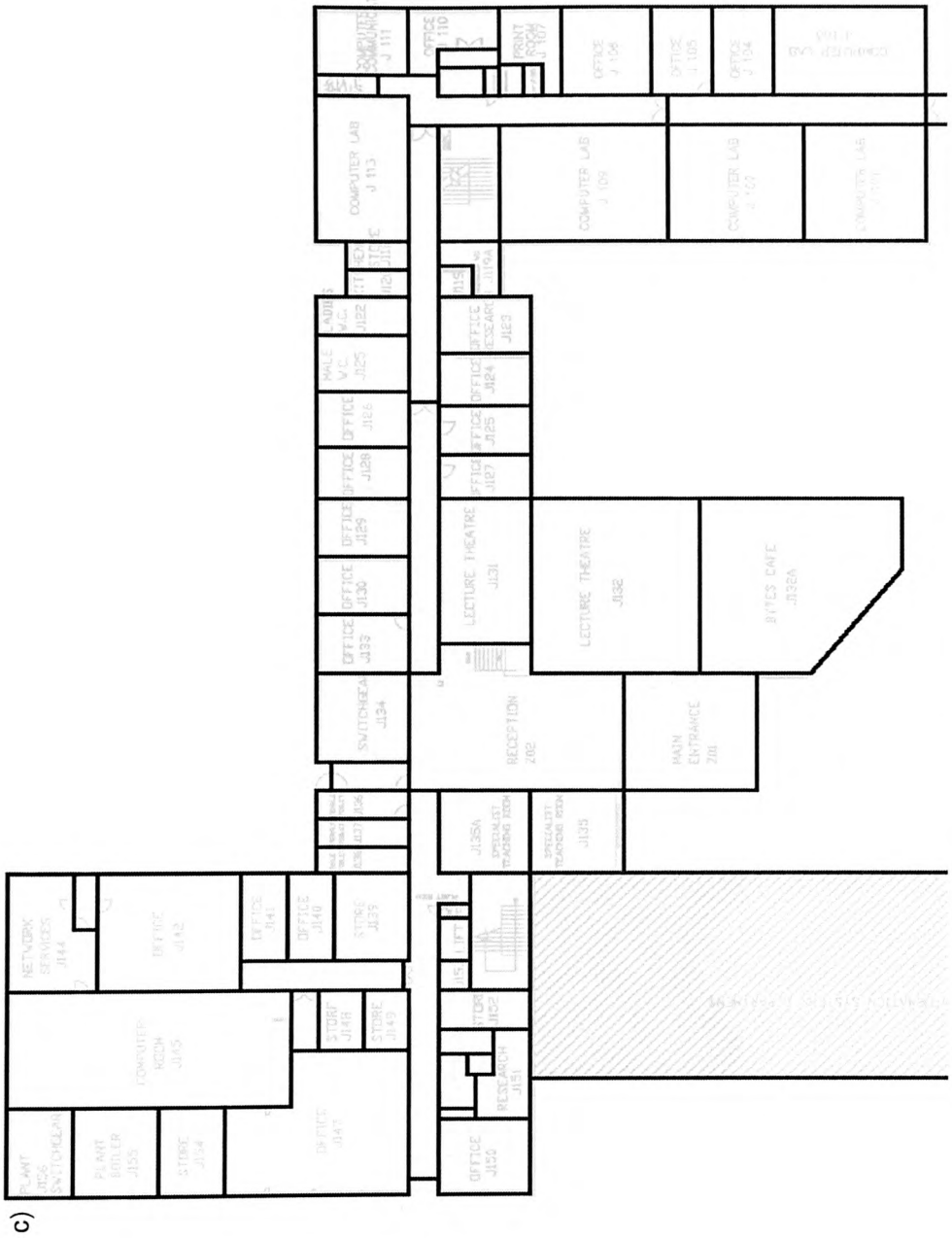

Figure 6.5 The model construction steps (H and J buildings from the University of Glamorgan campus): c) outlined rooms represented as polygons.

Figure 6.5 The model construction steps (H and J buildings from the University of Glamorgan campus): d) extruded rooms represented as polyhedra.

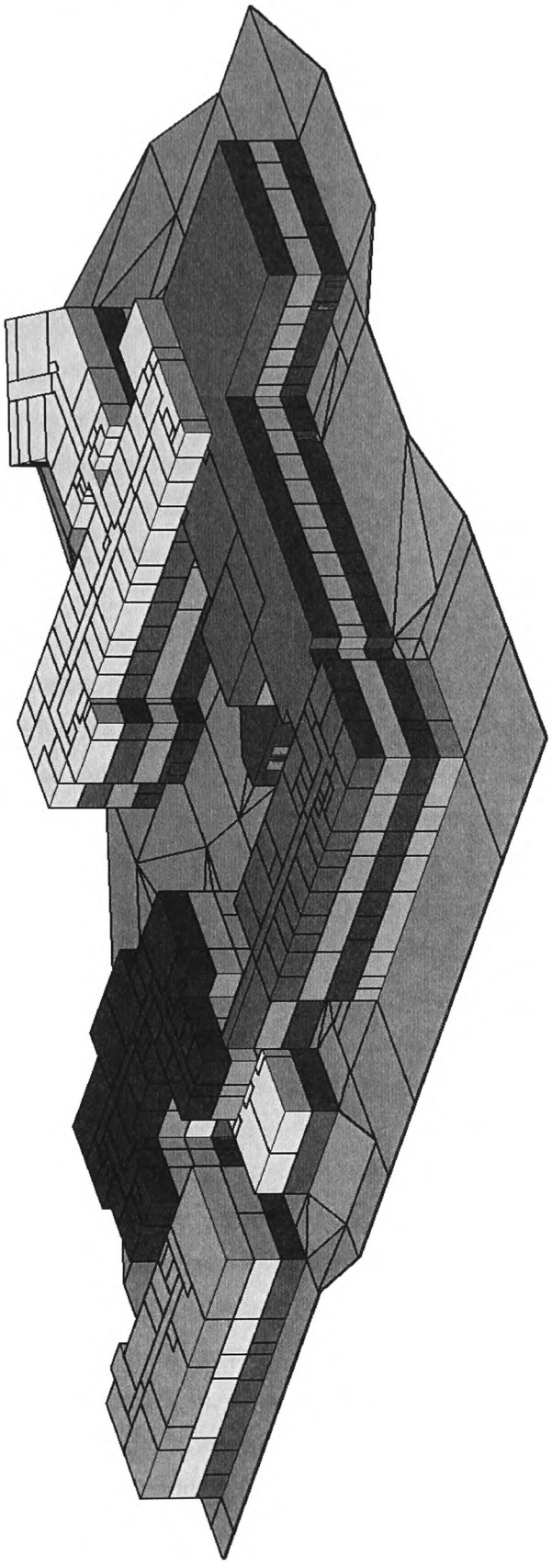

Figure 6.5 The model construction steps (H and J buildings from the University of Glamorgan campus): e) full model represented as a set of unconnected cells.

Big rooms or long corridors make indoor navigation more complex. An emergency situation (e.g. fire) in one part of a long corridor does not mean that all doors in the corridor are blocked and escape paths through this corridor are impossible. Decomposition of such rooms and corridors into a set of smaller connected cells makes the problem easier (Becker et al., 2009). A straight medial axis transformation proposed by Lee (2004) to create the geometric network model (see Section 2.7) may be applied. A different approach worth mentioning, which has lower combinatorial complexity than the medial axis (Aichholzer et al., 1995), is the straight skeleton proposed by Aichholzer and Aurenhammer (1996). Also, automatic partitioning of a building interior that is useful in pedestrian indoor navigation was presented by Stoffel et al. (2007). This method is based on the cell-and-portal decomposition (Lefebvre and Hornus, 2003) which uses cutting planes to subdivide space and can be applied to any architectural plan. The result is a set of cells connected by transparent 'portals'. This method can be supported by manually adding or removing portals in locations where the automatic method did not give satisfactory results.

In the proposed model partitioning of complex rooms is done manually. A 'portal' between two parts of the same room is called an 'open space' – this flat cell is used to connect two adjacent cells and has the same meaning in the weight calculation as the door element. 'Open spaces' were used to connect corridors split into smaller parts (vertical orientation) and also to connect staircases at subsequent floors (horizontal orientation).

A similar idea of 'open spaces' was applied by Meijers et al. (2005), where a semantic model for evacuation from buildings was presented. It is useful when big rooms (e.g. long corridors, shopping malls) need to be modelled. It is easier to split them into smaller pieces than to analyse one room of a complex shape. It may be necessary to assign attributes (that were assigned to the original room) to two or more cells representing the same room after the split. Because only a reference to an attribute is associated with a cell, it is not necessary to store attributes redundantly – only references need to be duplicated; an attribute itself is stored only once. The complex feature handling method described by Gröger and Plümer (2010b) may also be used. A hierarchical structure – a tree – is used for the maintenance of aggregated solids. Atomic solids (whole rooms or parts of a room) are stored in tree leaves, while the root node represents the complex object (e.g. a building). Attributes could be assigned to the nodes of the tree or directly to the solids represented by leaves. Thus attributes could be assigned to any feature at any level of aggregation, while the feature is not represented explicitly in a model – for example, to a building, floor, corridor, room, etc.

In the current example, building models represented by internal cells are enclosed by one external cell. Therefore all outside cells (at a boundary of a model) are connected to the exterior. However, it may be required to represent external space above the ground (air) and below the

ground (earth) as separate solids (Gröger and Plümer, 2010b). A tessellation of the air and ground surrounding a modelled object was also presented in a full 3D data model by Penninga (2008). In the campus building model, they would be represented as two separate cells with their dual volume vertices. This allows assigning different attributes not only to these volumes but also to connections between underground rooms and the earth cell, and to above-ground rooms and the air cell.

6.3 Escape routes: shortest path analysis

The shape of the campus building is complex and for that reason the path between two cells running entirely through the building interior is not always the shortest (see Figure 6.6a). Sometimes it is faster to find a shortcut outside the building (see Figure 6.6b), and emergency assembly points are usually located outside buildings. Thus the surrounding terrain is included in the model and should improve the efficiency of rescue simulations. Terrain is represented by cells (in the same manner as the building): they have a small thickness and they are connected to the building in the same way as building cells are connected together (see Figure 6.7). Thus the same graph traversal algorithms can be used for the terrain and building part. A schematic example is shown in Figure 6.8 – a projection of a ground floor with symbolic doors in a simple model of a building is shown. Using the same algorithm the shorter path between two rooms can be found when the exterior terrain is present in a model.

The Dijkstra algorithm (Dijkstra, 1959) was used on the dual graph to find the shortest path between two specific rooms. The same algorithm is used to find a route from a room to the nearest exit from a building (see Figure 6.9) – there is one source room and multiple exits. An exit can be any cell in a complex, but usually this is a cell representing a door connecting the building with the exterior. The locations of exits are not known at the beginning, thus they cannot be used as input parameter (which prevents the use of A^* instead of Dijkstra).

The weights used in the Dijkstra algorithm are assigned to the dual edges connecting adjacent rooms. They are calculated based on the geometrical distances between the two nodes representing adjacent rooms only if there is a door or 'open space' between these rooms. Otherwise the weight is not calculated (or can be considered infinite) and this connection is not taken into account in the search process. All temporary attributes (e.g. cumulative distance, previous edge on the path, etc.) are assigned to nodes and half-edges of the dual graph. After the path is found this is visualized as a highlighted set of primal cells that occur on the way from the source to the destination.

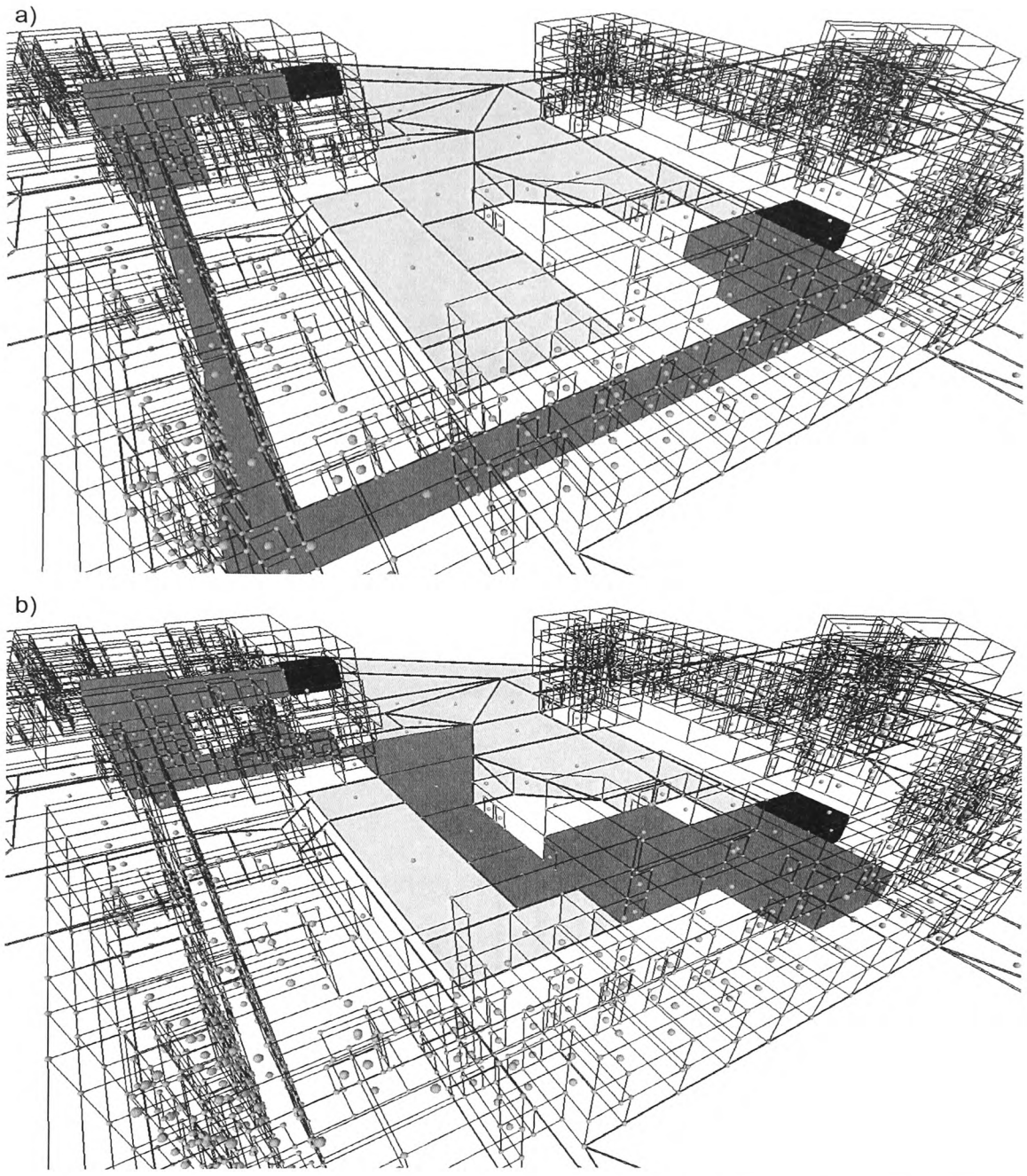

Figure 6.6 The shortest path (dark-grey) between two rooms (black) in the H&J building (external terrain represented by light-grey cells): a) the path entirely inside the building; b) the path calculated considering an external terrain.

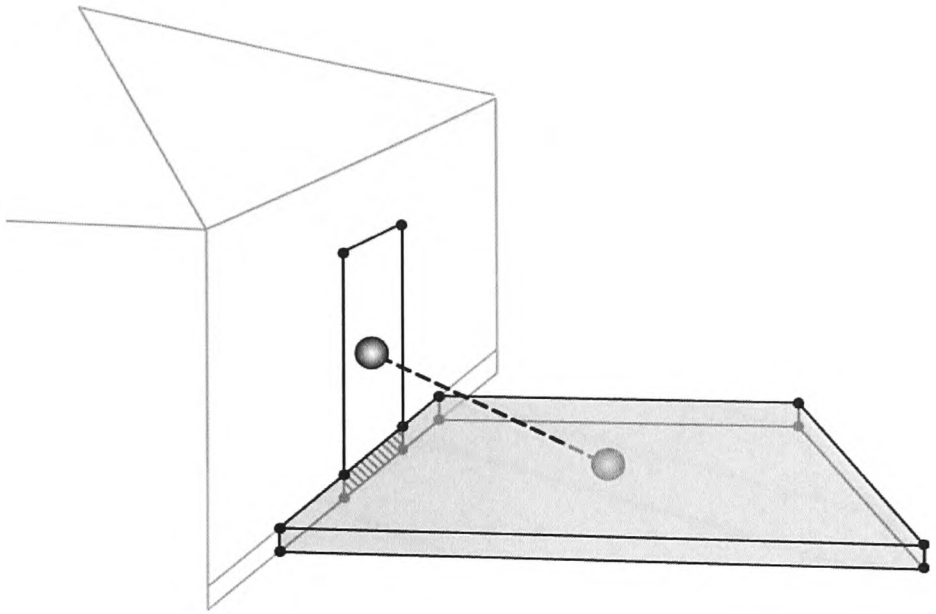

Figure 6.7 A navigable connection (a dual dashed line) between external terrain (grey thin cell) and building interior through a flat door; a door is linked with the terrain cell by a shared face (dashed area).

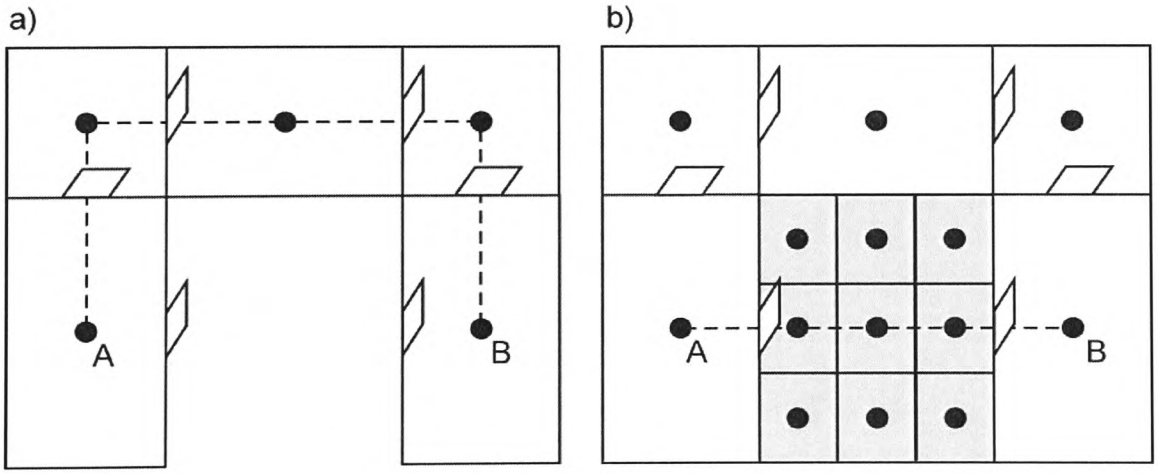

Figure 6.8 The shortest path (dotted line) between two rooms *A* and *B*: a) no exterior terrain – the path is entirely inside the building; b) the building with the exterior terrain (grey mesh) – a shorter path exists in the model.

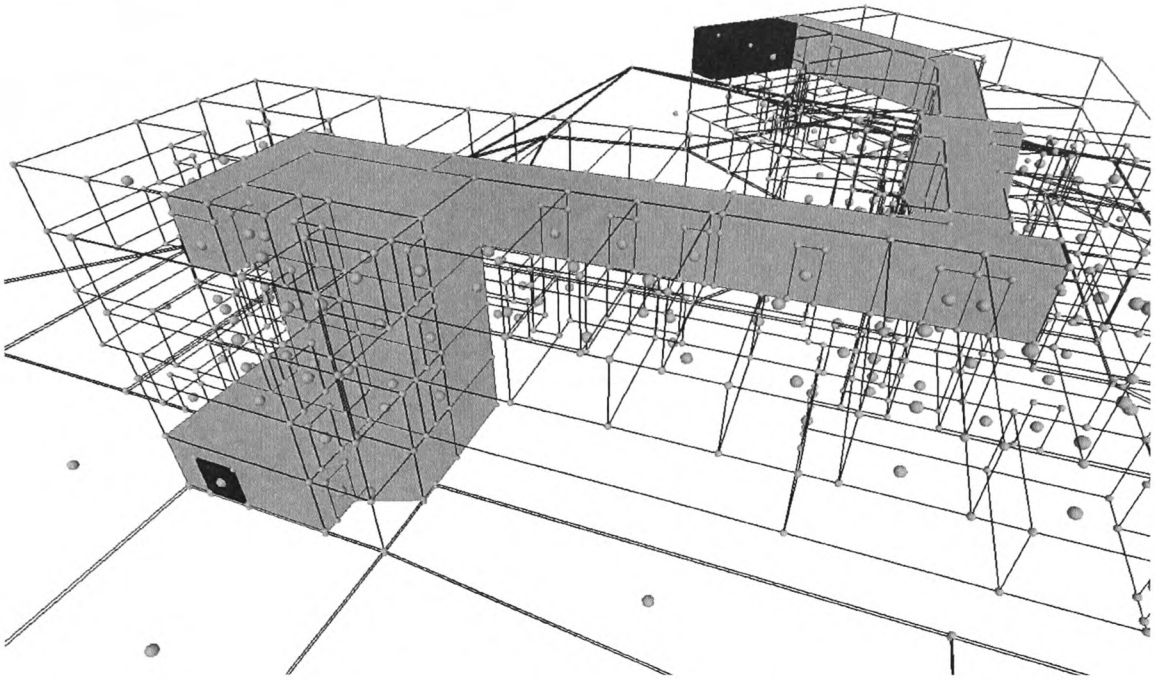

Figure 6.9 The shortest path between a room and the closest exit.

7 Conclusions

In this thesis, the DHE – a general data structure – was proposed. It can be used to represent the geometry and topology of a 3D digital spatial model with the special application of building interior modelling.

The representation of building interiors in 3D is quite problematic. Geometric models, for example created in CAD systems, are not adequate for further structure analysis because of missing topology. Adjacency relations between elements of a model are important to answer spatial queries. However adding topological information to the model increases the storage cost. This is even more problematic if non-manifold models need to be modelled. A data structure used for modelling should be simple and flexible to represent different kind of models. Also a standardized construction method is important because it can be used easily in different applications without a new analysis of data. The author believes the DHE data structure and the construction operators developed in this research are possible answers to that problem: other solutions may exist, but the proposed model handles the known mathematical cases and is practical for the analysis of real building models. In addition, its completeness is an asset where extensive analysis of attributes are required, such as directional traffic, the colours of the opposite faces of walls, or the properties of a particular room.

The fact that both the primal and dual subdivisions of a 3D model are explicitly stored and inter-connected has many advantages in practice; see for instance (Gold, 1991) for a discussion in 2D and (Ledoux and Gold, 2007) in 3D. It was demonstrated in Chapter 6 how the DHE and its construction and navigation operators are beneficial for the modelling and the analysis of 3D buildings, stored for instance in CityGML. An important benefit is that the dual is always constructed and preserved automatically, which allows not only for navigation between rooms but also for more complex operations, like routing in buildings, without having to reconstruct

the dual graph each time the model is modified, and to identify all the parts of a building that are accessible from a given starting location (for security purposes). Perhaps more importantly, the DHE is locally modifiable (i.e. without a global reconstruction of the structure) so that real-time modelling of buildings is possible; one can think of applications related to disaster management when a certain part of a building is inaccessible and the model must be updated and, consequently, the navigation network. Finally, the author believes the new data structure to be flexible and capable of handling common real-world situations such as holes in faces/polyhedra and non-manifold situations (e.g. when two polyhedra are only adjacent by one line segment or a vertex).

The comparison with other data structures presented in Sections 5.9.1 and 5.9.2 (the coupling-entity and the partial-entity) shows that the DHE is flexible and can be simplified to 'simulate' other solutions. The advantage of the DHE is that complex and non-manifold cases can be modelled. However it can be simplified to save storage space in a case of simple models which do not require a direct representation of a relationship between elements, and no complex spatial analysis is anticipated.

The comparison with other modelling method presented in Section 5.9.3 (3D NDM) shows that models created with the DHE are more general – they can be used in applications other than 3D building interior modelling. Dual connections representing the topology are constructed simultaneously with the geometry of the model, and all changes of the model are made locally. Thus they can be used in dynamically changing environment. DHE models may be further enriched with methods for the medial axis transformation and feature aggregation which allow for a similar representation as in the case of the 3D NDM.

In the following subsections some new ideas are presented. They would increase the functionality of the models developed during this research, and they may be a subject for future work.

7.1 Model storage in a relational data base

While the current implementation is graph-based, as objects in main memory, and only vertices and edges are required in the structure, only two tables are required to store the model in a relational data base. However a model may consist of many disconnected cell complexes thus links to the one of the edges from each independent complex are required. In addition a table storing attributes is needed if semantic information is included. To store the geometry and topology of a model consisting of many cell complexes together with attributes representing the semantic information of the model, four tables in a database are required (see Figure 7.1):

1. Complexes: (usually one) Separate cell complexes not connected together, but being a part of the same model.

2. Vertices: (containing coordinates) Primal vertices represent vertices in the primal space and volumes in the dual space. Dual vertices represent vertices in the dual space and volumes in the primal space. There will thus be pointers to both primal and dual attributes.
3. Edges: Primal edges represent edges in primal space and faces in dual space. Dual edges represent edges in the dual space and faces in the primal space. There will thus be pointers to both primal and dual attributes. It is important to remember that each undirected edge is stored in a table as two directed edges (half-edges).
4. Attributes: Attributes can be assigned to nodes and edges.

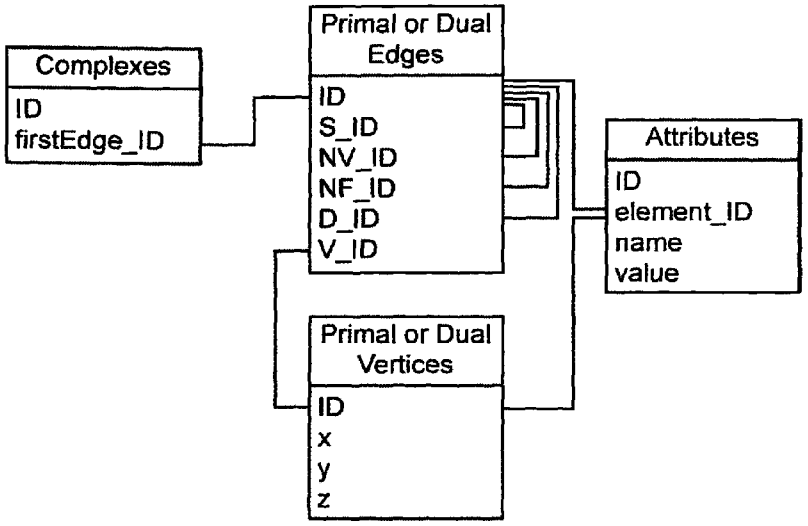

Figure 7.1 A simple idea of model storage in tables.

The idea of storing a model in a relational database was not developed in this project and needs further research.

7.2 Virtual museum

A complex building model was presented in Chapter 6. It should not be difficult to imagine the same model enriched with raster images of the interior – pictures of walls, floors, and ceilings attached to each face in the model would increase the realism of visualization. A graphic file with texture would be assigned as an attribute to a dual half-edge representing the appropriate side of a wall. In practice, a texture size and shape should fit the associated face, thus the texture must be prepared to accurately fit the face, or it must be scaled and adjusted to a face shape.

This development is an essential element of a possible application – a virtual museum.

The location of pictures, sculptures, and other works of art may be the same as in a real museum. People, who want to visit museums abroad but do not have the time or money to travel, could have a virtual walk, perhaps in a special presentation room with big screens. One

could define a field of interest to limit a number of rooms to visit, and focus on selected exhibitions.

Another possibility is to prepare a purely virtual museum – exhibit items would be virtually borrowed from different museums and located in one place. An exhibition might be put together according to user preferences or field of interest, and arranged into rooms.

Another issue, that needs improvement, is navigation inside a building. The meaning of navigation here is different from navigation in a graph structure used for modelling (this is well defined). An issue is intuitive navigation in a virtual building, for example: walking through walls and ceilings is not allowed (only doors can be used to walk from one room to another); an eye level (an eye distance from a floor) should be preserved to avoid 'flying' over a floor. These rules could be taken from 3D computer game engines.

Shopping centre assistance is another related idea. Customers visiting big shopping malls could plan their shopping. One could choose shops to visit and give some preferences (e.g. preferred visit order, preferred brands, a restriction on visited shops, etc.). The shortest/fastest walking route accompanied with shop-face textures and other landmarks would be displayed, printed, or sent to a mobile device. This idea may be also applied to information centres in big buildings or building complexes to find an office, room, or other location (e.g. government buildings, hospitals, university campuses, etc.).

7.3 Training simulators

The virtual museum is only a simple example. This can be enriched with simulations of different phenomena (i.e. fire, smoke, noise propagation in a building; a bomb explosion). Such models could be used in simulators to train firemen, soldiers, anti-terrorist squads, etc. and prepare them for disasters in the real world – instead of walking around a museum and looking at walls they could take decisions and interact: this door is on fire; let's make an alternative route through that wall; how thick is the wall? is it made of concrete? can we make hole with an axe?; are there people trapped in the next room? A system could answer some of these questions and help them to make the right decision.

A model used in a disaster management system, where changes of the environment can be frequent and fast, cannot be static: the structure of a building may be changed by explosions; corridors may be blocked by fire; additional passages may be created (e.g. ladders used by firemen). To allow dynamic changes in the model an interface to a quick model modification module should be developed. For example, operations that should be possible in the application might be: to block doors by pointing them out in the model, or using a door ID or number; to block a passage using a selected corridor or room; etc. (Dynamic and local changes of the model are possible using operators described in this thesis: only the interface needs development.)

Dynamic changes of the model may result in a blockage of the shortest paths calculated beforehand (for a static model, which was not affected by the disaster). Thus new modified evacuation routes needs to be calculated. To avoid recalculation of all routes from the beginning (for example, using the Dijkstra algorithm) a modified algorithm taking into consideration changes of the existing paths should be used, for example as proposed by Musliman et al. (2008).

7.4 Building interior surveying

The availability of an incrementally constructed non-manifold model may open up a set of applications not previously considered. One example is interior building surveying – a particularly difficult task in the absence of indoor GPS. The availability of relatively simple equipment for determining the position of some (non-reflective) point from a reference position permits rapid two-person surveying: one to take the reading and the other to express its relation to previous points (for example the next corner around a ceiling). Equipment currently available has not the high precision of complete surveying systems, but it appears to be sufficient to express the relationships between adjacent rooms – the initial impetus for this project.

A process of a building interior surveying would start with marking out a base point located outside the building and determined using GPS equipment. This point should be located close to any entrance (i.e. door, window) to be visible from inside. Building interior measurements will be relative to the base point, as GPS cannot be used inside buildings. Next base points set inside the building are necessary to determine a relative position to the original base point.

The surveying of a single room should be started from defining a new room object (represented as a dual node). Then, face by face a room shape is reconstructed. Each face is 'stretched' by adding new corners preserving their order around the face (i.e. clockwise or anti-clockwise); all measurements are taken from inside of a room. The object is complete if all the faces form a closed shell – this can be easily detected using the DHE. A cell constructed in this way is not linked with the rest of the building model; this can be done automatically based on the geometry.

In this surveying method the equipment accuracy should not exceed a few centimetres; otherwise adjacent cells representing rooms could overlap (especially in a case of thin walls and partitions); thus an overlapping detection tool is necessary.

Walls are another element that should be included in a model to precisely map a building interior into a model – walls have thickness, there may be installations inside walls. Adjacent rooms would not be linked directly, but through the walls (like in the 'Type 1' CityGML model example described in Chapter 6). This does not affect the indoor navigation, because openings like doors and windows are also modelled – navigation is possible with these objects. A model

without thick walls (the 'Type 2' model – the building from the University of Glamorgan campus described in Chapter 6) could be obtained from the original one, but rooms would have to be 'stretched' in order to fill gaps between these rooms. Dimension reduction of solid models using mid-surfaces was described by Sheen et al. (2007): thick parallel walls are exchanged for flat faces.

Error in measurements (caused by equipment imprecision) will increase with the distance from the original base point. Thus, to minimize this effect, several base points located outside the building should be set out if possible; or an external measurement of the building may be used to compare with the total size of the rooms (for example, to avoid rooms outside building boundaries). A technique to lay out rooms inside building boundaries, and correcting the measurement error, needs to be developed.

In buildings, especially in historical houses, it is possible to find hidden or secret rooms. They were used to hide people or valuables. The described surveying application could help to find such rooms – too thick walls, not utilized space under the stairs would be suspicious.

7.5 Other applications

Further applications, perhaps within BIM, CityGML or other full 3D applications are clearly possible, and await discovery. For example, Gröger and Plümer (2010b) show how to decompose a 3D city model into simple solids with topological consistency checking rules. However, no particular data structure is suggested. The author believes that the DHE data structure, which stores topological connections between elements, could be a useful supplement to their work. Also the modelling rules described there fit with the construction principles presented in this thesis.

The author believes the DHE fits into the FEM method – cell complexes representing tessellated objects could be used in this method. An advantage of having dual graphs at the same time has significant meaning, for example for two- and three-dimensional electromagnetic simulation (Sazanov et al., 2007). Similar methods would apply for noise and radio signal propagation, among others.

References

Aichholzer, O. and Aurenhammer, F., 1996. Straight skeletons for general polygonal figures in the plane. Computing and Combinatorics: 117-126.

Aichholzer, O., Aurenhammer, F., Alberts, D. and Gärtner, B., 1995. A Novel Type of Skeleton for Polygons. Journal of Universal Computer Science, 1(12): 752-761.

Albrecht, J.H., 1996. Universal GIS operations. PhD Thesis, ISPA - University of Vechta, Vechta, Germany.

Banks, J., Carson, J., Nelson, B.L. and Nicol, D., 2009. Discrete-Event System Simulation. Prentice Hall.

Baumgart, B., G., 1975. A polyhedron representation for computer vision, Proceedings of the May 19-22, 1975, national computer conference and exposition. ACM, Anaheim, California.

Becker, T., Nagel, C. and Kolbe, T.H., 2009. A Multilayered space-event model for navigation in indoor spaces. 3D Geo-Information Sciences: 61-77.

Benner, J., Geiger, A. and Leinemann, K., 2005. Flexible generation of semantic 3D building models.

Berg, M., Cheong, O., Kreveld, M. and Overmars, M., 2008. Computational Geometry: Algorithms and Applications. Springer.

Boguslawski, P. and Gold, C., 2008. Construction Operators for Modelling 3D Objects. In: P. Plassmann and P. Roach (Editors), 3rd Research Student Workshop, University of Glamorgan, UK, pp. 70-74.

Boguslawski, P. and Gold, C., 2009a. Construction operators for modelling 3D objects and dual navigation structures. In: J. Lee and S. Zlatanova (Editors), 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, pp. 47-59.

Boguslawski, P. and Gold, C., 2009b. Unified 2D Terrain and 3D Multi-Shell Building Models, ISPRS International Workshop on Multidimensional & Mobile Data Model, UTM Johor, Malaysia.

Boguslawski, P. and Gold, C., 2010. Euler Operators and Navigation of Multi-shell Building Models. In: T. Neutens and P. Maeyer (Editors), Developments in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, pp. 1-16.

Boguslawski, P. and Gold, C., 2011. Rapid Modelling of Complex Building Interiors. In: T.H. Kolbe, G. König and C. Nagel (Editors), Advances in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography, pp. 43-56.

Boguslawski, P., Gold, C.M. and Ledoux, H., 2011. Modelling and analysing 3D buildings with a primal/dual data structure. ISPRS Journal of Photogrammetry and Remote Sensing, 66(2): 188-197.

Braid, I.C., Hillyard, I.C. and Stroud, I.A., 1980. Stepwise Construction of Polyhedra in Geometric Modelling. In: e. K.W.Brodlie (Editor), Mathematical Methods in Computer Graphics and Design. Academic Press, pp. 123-141.

Brisson, E., 1993. Representing Geometric Structures in d Dimensions: Topology and Order. Discrete & Computational Geometry, 9: 387-426.

Burrough, P.A. and McDonnell, R.A., 1998. Principles of Geographical Information Systems. Oxford University Press, Oxford, 327 pp.

Choi, J. and Lee, J., 2009. 3D Geo-Network for Agent-based Building Evacuation Simulation. In: J. Lee and S. Zlatanova (Editors), 3D Geo-Information Sciences. Springer, pp. 283-299.

Coors, V., 2003. 3D-GIS in networking environments. Computers, Environment and Urban Systems, 27(4): 345-357.

Coors, V., 2008. On the convergence of 3D-GIS, CAD and 3D Simulation, 7th International Conference & Exhibition on Geo-information (ISG), Kuala Lumpur, Malaysia.

Coors, V. and Flick, S., 1998. Integrating Levels of Detail in a Web-based 3D-GIS. ACM, pp. 40-45.

Coors, V. and Rossignac, J., 2004. Delphi: geometry-based connectivity prediction in triangle mesh compression. The Visual Computer, 20(8): 507-520.

Coppock, J.T. and Rhind, D.W., 1991. The history of GIS. Geographical information systems: Principles and applications, 1: 21–43.

Couclelis, H., 1992. People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS, Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, pp. 65-77.

De Kemp, E., 2007. 3-DGeological Modelling Supporting Mineral Exploration. Mineral deposits of Canada: a synthesis of major deposit types, district metallogeny, the evolution of geological provinces, and exploration methods. – Spec. Publ., Geol. Assoc. Canada, Mineral Deposits Division, 5: 1051-1061.

Delaunay, B., 1934. Sur la sphère vide. Bulletin of Academy of Sciences of the USSR(6): 793-800.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik, 1: 269-271.

Dirichlet, G.L., 1850. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. Journal für die Reine und Angewandte Mathematik, 40: 209-227.

Dobkin, D.P. and Laszlo, M.J., 1987. Primitives for the manipulation of three-dimensional subdivisions, Proceedings of the third annual symposium on Computational geometry. ACM, Waterloo, Ontario, Canada.

Egenhofer, M.J., 1995. Topological relations in 3D, Technical report, University of Maine, USA.

Ellul, C., 2007. Functionality and Performance - Two Important Considerations when Implementing Topology in 3D. PhD Thesis, University College London.

Ellul, C. and Haklay, M., 2006. Requirements for topology in 3D GIS. Transactions in GIS, 10(2): 157-175.

Filippoupolitis, A., Loukas, G., Timotheou, S., Dimakis, N. and Gelenbe, E., 2009. Emergency Response Systems for Disaster Management in Buildings, pp. 11–12.

Gold, C. and Angel, P., 2006. Voronoi hierarchies. In: M. Raubal, H. Miller, A. Frank and M. Goodchild (Editors), Geographic, Information Science. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 99-111.

Gold, C., M., 2006. What is GIS and What is Not? Transactions in GIS, 10(4): 505-519.

Gold, C.M., 1987. Spatial ordering of Voronoi networks and their use in terrain data base management. In: N.R. Chrisman (Editor), Auto-Carto 8, Baltimore, MD, USA, pp. 185-194.

Gold, C.M., 1991. Problems with handling spatial data - the Voronoi approach. CISM Journal, 45(1): 65-80.

Gold, C.M., 2005. Data structures for dynamic and multidimensional GIS, 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS, Pontypridd, Wales, UK, pp. 36-41.

Gold, C.M., Tse, R.O.C. and Ledoux, H., 2006. Building Reconstruction - Outside and In. In: A.A. Rahman, S. Zlatanova and V. Coors (Editors), Innovations in 3D Geo Information Systems. Lecture Notes in Geoinformation and Cartography. Springer, pp. 355-369.

Goodchild, M.F., 1987. Spatial analytical perspective on geographical information systems. International Journal of Geographical Information Systems, 1(4): 327–334.

Goodchild, M.F., 2006. GIS and disasters: Planning for catastrophe. Computers, Environment and Urban Systems, 30: 227-229.

Gröger, G. and Plümer, L., 2010a. Derivation of 3D Indoor Models by Grammars for Route Planning. 191-206.

Gröger, G. and Plümer, L., 2010b. How to achieve consistency for 3D city models. Geoinformatica, 15(1): 137-165.

Grünbaum, B. and Shephard, G.C., 1986. Tilings and patterns. W. H. Freeman & Co., New York, 700 pp.

Guibas, L. and Stolfi, J., 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi Diagrams. ACM Trans. Graph., 4(2): 74-123.

Isikdag, U. and Zlatanova, S., 2009. Towards Defining a Framework for Automatic Generation of Buildings in CityGML Using Building Information Models. In: J. Lee and S. Zlatanova (Editors), 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer.

Isikdag, U. and Zlatanova, S., 2010. Interactive modelling of buildings in Google Earth: A 3D tool for Urban Planning. In: T. Neutens and P. de maeyer (Editors), Developments in 3D Geo-Information Sciences, pp. 52-70.

ISO/PAS 16739, 2005. Industry Foundation Classes, Release 2x, Platform Specification (IFC2x Platform).

Kolbe, T.H., 2009. Representing and Exchanging 3D City Models with CityGML. In: J. Lee and S. Zlatanova (Editors), 3D Geo-Information Sciences. Springer Berlin Heidelberg, pp. 15-31.

Kolbe, T.H., Gröger, G. and Plümer, L., 2008. CityGML - 3D city models and their potential for emergency response. In: S. Zlatanova and J. Li (Editors), Geo-Information technology for emergency response. Taylor&Francis.

Kwan, M.-P. and Lee, J., 2005. Emergency response after 9/11: the potential of real-time 3D GIS for quick emergency response in micro-spatial environments. Computers, Environment and Urban Systems, 29: 93-113.

Ledoux, H., 2006. Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual. PhD Thesis, University of Glamorgan, 204 pp.

Ledoux, H. and Gold, C.M., 2007. Simultaneous storage of primal and dual three-dimensional subdivisions. Computers, Environment and Urban Systems, 31(4): 393-408.

Ledoux, H. and Meijers, M., 2010. Topologically consistent 3D city models obtained by extrusion. International Journal of Geographical Information Science, In Press.

Ledoux, H., Verbree, E. and Hand, S., 2009. Geometric Validation of GML Solids with the Constrained Delaunay Tetrahedralization. In: P. De Maeyer, T. Neutens and M. De Ryck (Editors), 4th International Workshop on 3D Geo-Information, Ghent, Belgium, pp. 143-148.

Lee, J., 2004. A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities. Geoinformatica, 8(3): 237-264.

Lee, J., 2007. A Three-Dimensional Navigable Data Model to Support Emergency Response in Microspatial Built-Environments. Annals of the Association of American Geographers, 97(3): 512 - 529.

Lee, J. and Kwan, M.P., 2005. A combinatorial data model for representing topological relations among 3D geographical features in micro-spatial environments. International Journal of Geographical Information Science, 19(10): 1039 - 1056.

Lee, J. and Zlatanova, S., 2008. A 3D data model and topological analyses for emergency response in urban areas. In: S. Zlatanova and J. Lee (Editors), Geospatial Information Technology for Emergency Response. Taylor & Francis, pp. 143-168.

Lee, K., 1999. Principles of CAD/CAM/CAE Systems. Addison-Wesley Longman Publishing Co., Inc., 582 pp.

Lee, S.H. and Lee, K., 2001. Partial entity structure: a compact non-manifold boundary representation based on partial topological entities, Proceedings of the sixth ACM

symposium on Solid modeling and applications. ACM, Ann Arbor, Michigan, United States.
Lefebvre, S. and Hornus, S., 2003. Automatic cell-and-portal decomposition, Institut National de Recherche en Informatique et en Automatique.
Li, Z., Zhu, Q. and Gold, C., 2004. Digital Terrain Modeling: Principles and Methodology. CRC Press, 323 pp.
Lienhardt, P., 1991. Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer Aided Design, 23(1): 59-82.
Lipson, H. and Shpitalni, M., 1998. On the Topology of Sheet Metal Parts. ASME Journal of Mechanical Design, 120(1): 10-16.
Mallet, J.-L., 2002. Geomodeling. Oxford University Press.
Mäntylä, M., 1988. Introduction to Solid Modeling. Computer Science Press, Inc., 401 pp.
Masuda, H., 1993. Topological operators and Boolean operations for complex-based nonmanifold geometric models. Computer-Aided Design, 25(2): 119-129.
Masuda, H., Shimada, K., Numao, M. and Kawabe, S., 1989. A Mathematical Theory and Applications of Non-Manifold Geometric Modeling International Symposium on Advanced Geometric Modeling for Engineering Applications, Berlin, Germany, pp. 89-103.
McGaughey, J., 2006. The Common Earth Model: A Revolution in Mineral Exploration Data Integration. GIS for the Earth Sciences: Geological Association of Canada, Special Publication, 44: 567-576.
Meijers, M., Zlatanova, S. and Pfeifer, N., 2005. 3D Geo-Information Indoors: Structuring for Evacuation, Next Generation 3D City Models, Bonn, Germany, pp. 6.
Molenaar, M., 1992. A topology for 3 D vector maps. ITC Journal(1): 25-33.
Munkres, J.R., 1984. Elements of Algebraic Topology. Addison-Wesley Publishing Company, Inc.
Musliman, I.A., Rahman, A.A. and Coors, V., 2008. Implementing 3d network analysis in 3D-GIS. Proceedings of XXI International Society for Photogrammetry and Remote Sensing, Beijing, China. Volume , Part B, 2.
OGC, 2007. Geography Markup Language (GML) Encoding Standard. Open Geospatial Consortium Inc.
OGC, 2008. City Geography Markup Language (CityGML) Encoding Standard. Open Geospatial Consortium Inc.
Penninga, F., 2008. 3D topography : a simplicial complex-based solution in a spatial DBMS. PhD Thesis, Delft University of Technology, Delft, 192 pp.
Pilouk, M., 1996. Integrated modelling for 3D GIS. PhD Thesis, ITC, The Netherlands.
Pu, S. and Zlatanova, S., 2005. Evacuation Route Calculation of Inner Buildings. In: P.J.M. van Oosterom, S. zlatanova and E.M. Fendel (Editors), Geo-information for disaster management. Springer Verlag, Heidelberg, pp. 1143-1161.
Rossignac, J. and Cardoze, D., 1999. Matchmaker: manifold BReps for non-manifold r-sets, Proceedings of the fifth ACM symposium on Solid modeling and applications. ACM, Ann Arbor, Michigan, United States.
Rossignac, J., R. and O'Connor, M., A., 1990. SGC: A dimension-independent model for point-sets with internal structures and incomplete boundaries. In: M. Wozny, J., J. Turner, U. and K. Preiss (Editors), Geometric Modeling for Product Engineering. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, pp. 145-180.
Rossignac, J.R. and Requicha, A.A.G., 1999. Solid modeling. In: J.W.S. Webster (Editor), Encyclopedia of Electrical and Electronics Engineering.
Sazanov, I., Hassan, O., Morgan, K. and Weatherill, N.P., 2007. Generating the Voronoi–Delaunay Dual Diagram for Co-Volume Integration Schemes, The 4th International Symposium on Voronoi Diagrams in Science and Engineering 2007 (ISVD 2007), pp. 199-204.
Schneider, M., 2009. Spatial Data Types. In: L. Liu and M.T. Özsu (Editors), Encyclopedia of Database Systems. Springer-Verlag.

Schulte, C. and Coors, V., 2008. Development of a CityGML ADE for dynamic 3D flood information, Joint ISCRAM-CHINA and GI4DM Conference on Information Systems for Crisis Management, Harbin, China.

Shapiro, V., 2002. Solid Modeling. In: J.H. G. Farin, M.-S. Kim, eds. (Editor), Handbook of Computer Aided Geometric Design. Elsevier Science Publishers, pp. 473-518.

Sheen, D.-P., Son, T.-g., Ryu, C., Lee, S.H. and Lee, K., 2007. Dimension Reduction of Solid Models by Mid-Surface Generation. International Journal of CAD/CAM, 7(1): 71-80.

Slingsby, A. and Raper, J., 2008. Navigable Space in 3D City Models for Pedestrians. In: P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Editors), Advances in 3D Geoinformation Systems. Lecture Notes in Geoinformation and Cartography. Springer.

Smith, B., 1995. On drawing lines on a map, Spatial Information Theory A Theoretical Basis for GIS, pp. 475-484.

Stefanakis, E. and Patroumpas, K., 2008. Google Earth and XML: Advanced Visualization and Publishing of Geographic Information. In: M.P. Peterson (Editor), International Perspectives on Maps and the Internet. Springer, pp. 143-152.

Stoffel, E.P., Lorenz, B. and Ohlbach, H.J., 2007. Towards a Semantic Spatial Model for Pedestrian Indoor Navigation Advances in Conceptual Modeling – Foundations and Applications. Lecture Notes in Computer Science, pp. 328-337.

Stoter, J. and Zlatanova, S., 2003. 3D GIS, where are we standing, Joint Workshop on Spatial, Temporal and Multi-Dimensional Data Modeling and Analysis, Quebec city, Canada.

Stroud, I., 2006. Boundary Representation Modelling Techniques. Springer-Verlag New York, Inc.

The CGAL Project, 2010. CGAL User and Reference Manual. CGAL Editorial Board.

Thomsen, A., Breunig, M., Butwilowski, E. and Broscheit, B., 2008. Modelling and Managing Topology in 3D Geoinformation Systems. In: P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Editors), Advances in 3D Geoinformation Systems. Lecture Notes in Geoinformation and Cartography. Springer.

Tse, R., O. C. and Gold, C., M., 2004. TIN meets CAD: extending the TIN concept in GIS. Future Gener. Comput. Syst., 20(7): 1171-1184.

Tse, R.O.C., Gold, C. and Kidner, D., 2008. 3D City Modelling from LIDAR Data, Advances in 3D Geoinformation Systems, pp. 161-175.

van Oosterom, P., Stoter, J., Quak, W. and Zlatanova, S., 2002. The balance between geometry and topology, 10th Int. Symposium on Spatial Data Handling. Springer-Verlag, pp. 209-224.

Voronoi, G., 1908. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Journal für die reine und angewandte Mathematik, 1908(134): 198-287.

Wegener, M., 2001. New spatial planning models. International Journal of Applied Earth Observation and Geoinformation, 3(3): 224-237.

Weiler, K., 1988. The Radial Edge data structure: A topological representation for non-manifold geometric boundary modeling. In: J. Encarnacao, L., M. Wozny, J. and H. McLaughlin, W. (Editors), Geometric Modeling for CAD Applications. Elsevier Science (North-Holland), Amsterdam, pp. 3-36.

Weisstein, E.W., 1999. Graph (http://mathworld.wolfram.com/Graph.html). Wolfram Research, Inc.

Weisstein, E.W., 2005. Graph (http://mathworld.wolfram.com/Graph.html). Wolfram Research, Inc.

Worboys, M. and Duckham, M., 2004. GIS: A Computing Perspective. CRC Press.

Yamaguchi, Y. and Kimura, F., 1995. Nonmanifold Topology Based on Coupling Entities. IEEE Comput. Graph. Appl., 15(1): 42-50.

Yamaguchi, Y., Kobayashi, K. and Kimura, F., 1991. Geometric Modeling with Generalized Topology and Geometry for Product Engineering. In: J.P.a.M.W.E. J. Turner (Editor), Product Modeling for Computer-Aided Design and Manufacturing. Elsevier Science Publishers B.V. (North-Holland).

Yanbing, W., Lixin, W., Wenzhong, S. and Xiaomeng, L., 2007. On 3D GIS spatial modeling, ISPRS Workshop on Updating Geo-spatial Databases with Imagery & The 5th ISPRS Workshop on DMGISs, pp. 237-240.

Zlatanova, S., 2000a. 3D GIS for Urban Development. PhD Thesis, Graz University of Technology.

Zlatanova, S., 2000b. On 3D Topological Relationships, 11th International Workshop on Database and Expert Systems Applications (DEXA'00), pp. 913.

Zlatanova, S., Rahman, A.A. and Shi, W., 2004. Topological models and frameworks for 3D spatial objects. Computers & Geosciences, 30(4): 419-428.